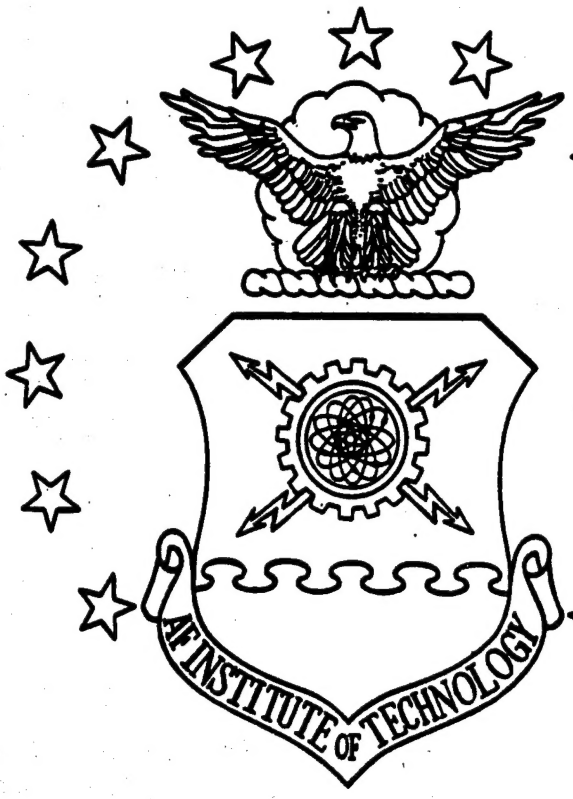

EXMIS: AN EXPERT SYSTEM FOR THE AUTONOMOUS ONBOARD COMMAND SYSTEM (AOCS

THESIS

Kevin D. Benedict, Captain, USAF

AFIT/GSO/ENY/95D-01

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GSO/ENY/95D-01

EXMIS: AN EXPERT SYSTEM FOR THE AUTONOMOUS ONBOARD COMMAND SYSTEM (AOCS)

THESIS

Kevin D. Benedict, Captain, USAF

AFIT/GSO/ENY/95D-01

Approved for public release; distribution unlimited

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense of the U. S. Government

AFIT/GSO/ENY/95D-01

EXMIS: AN EXPERT SYSTEM FOR THE AUTONOMOUS ONBOARD COMMAND SYSTEM (AOCS)

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air Education and Training Command

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Space Operations

Kevin D. Benedict, B.S., M.A.

Captain, USAF

December 1995

Acknowledgments

I would like to take this opportunity to thank the many people who have helped me through one of the hardest and most rewarding endeavors I have ever undertaken.

First, I must thank Capt David Gill and 1st Lt Sean Oswalt from the Ballistic and Space Design Branch (TANB) at the National Air Intelligence Center (NAIC). Your assistance and answers to my many questions were a tremendous help with my research project. I know there were better things you could have done with your time.

Next, I would like to thank my thesis advisor Dr. Chris Hall. Our stimulating conversations and your supportive answers gave me the right amount of assistance when I needed it.

I would also like to thank the project sponsor, Mr. Fred Foerst, for providing a great opportunity to a struggling graduate student. Your willingness to take time out of your busy schedule was both admirable and greatly appreciated.

Last, but certainly not least, I must thank the four people who have sacrificed the most through this whole ordeal. My children, Cassandra, Tasia, and Alexandria, who loved daddy even when his temper tantrums were both loud and frequent, and my best friend, (who also happens to be my wife) Pam. You are always there for me through better or worse. May the better begin now.

Table of contents

List of Figures

List of Tables

List of Terms

ACS - Attitude Control System

AI - Artificial Intelligence

AOCS - Autonomous Onboard Command System

APEX - Autonomous Power Expert

CLIPS - **C** Language **I**ntegrated **P**roduction **S**ystem

EOM - Equations of Motion

ER - Eastern Range

EXMIS - **EX**pert **MI**ssile System

FCO - Flight Control Officer

GN - Ground Network

GPS - Global Positioning System

IG - Inertial Guidance

IMU - Inertial Measurement Unit

ISTAR - Intelligent System for Telemetry Analysis in Real-time

KBS - Knowledge Based Systems

LHS - left-hand side

LISP - *LIS*t *P*rocessing language

LRS - Launch Range System

MFCO - Mission Flight Control Officers

NAIC - National Air Intelligence Center

PC Plus - Personal Consultant Plus

RE - Ranging Equipment

RIM - Ranging Internal Monitor

RSA - Range Standardization and Automation

SMART - Spacecraft Monitoring And Real-time Telemetry

TANB - Ballistic and Space Design Branch

TDRS - Tracking and Data Relay Satellite

WR - Western Range

Abstract

The primary objective of this study is the creation of a prototype expert system called EXMIS (**EX**pert **MI**ssile **S**ystem) that performs the flight termination logic sequence of the Autonomous Onboard Command System (AOCS). The AOCS is a proposed system that would take the "man-in-the-loop" out of the self-destruct decision making process and place the entire decision on the launch vehicle. Through the use of a six degree of freedom trajectory program called BOOST, simulated flight data for four different flight scenarios is obtained. The launch vehicle selected for the simulations is the Peacekeeper ICBM.

EXMIS is developed using an expert system shell called CLIPS (**C** Language **I**nte-grated **P**roduction **S**ystem) designed at the NASA/Johnson Space Center. CLIPS is a for-ward chaining rule-based language that has inferencing and representation capabilities. Once developed, the BOOST simulation data are used to evaluate EXMIS under different scenarios involving nominal, errant, and unstable launch vehicle flight.

A recommendation is made for further testing of the prototype EXMIS system and to pur-sue development of an advanced EXMIS program with the assistance of an expert in the AOCS area.

EXMIS: AN EXPERT SYSTEM FOR THE AUTONOMOUS ONBOARD COMMAND SYSTEM (AOCS)

I. Introduction

The Launch Range System (LRS), which includes the Eastern Range (ER) at Cape Canaveral, Florida and the Western Range (WR) at Vandenberg AFB, California, has been undergoing a Range Standardization and Automation (RSA) program to modernize, standardize, and automate both ranges (3). The Autonomous Onboard Command System (AOCS) is a proposed system that would be a key part of this modernization effort. It would effectively remove the "Man-In-The-Loop" from launch termination decisions and replace it with an autonomous system located onboard the launch vehicle. As described in the Spacelift Range Architecture Study , "The AOCS concept is to use the onboard navigation and status, onboard GPS (Global Positioning System), possible optional inertial sensors, and microprocessors with stored destruct criteria to implement real-time range safety functions." (9).

The main focus of this research effort was to evaluate the feasibility of using an expert system to perform the flight termination logic for the AOCS should the launch vehicle

violate established pre-launch safety parameters. This task was accomplished through several separate steps. First, simulated launch data were obtained using the BOOST program developed by Martin Marietta Corporation. Second, an expert system dubbed EXMIS (for EXpert MIssile System) was created through the use of the expert system language CLIPS (C Language Integrated Production System) developed by NASA/Johnson Space Center. Third, the launch data are processed by EXMIS to determine if the expert system can recognize when pre-launch safety criteria have been violated and perform the appropriate actions (e.g. self destruct). Four different scenarios were created in order to test a range of possible realistic events.

Chapter 1 provides a background of current procedures, the AOCS, and the approach of this research. Chapter 2 looks at the related efforts in the field of Expert Systems and how they are being used in space applications. Chapter 3 develops the equations of motion for a rocket in three dimensions and focuses primarily on the BOOST program and how it was used to obtain simulated launch data. Chapter 4 describes the EXMIS expert system and how it was developed. Chapter 5 gives the results of testing EXMIS with the BOOST data. Finally, Chapter 6 presents conclusions and recommendations for future work. Appendices are provided for the entire EXMIS program and a list of variables used in the BOOST program. It should be pointed out from the start that this research effort is concerned with designing an expert system for the command logic portion of the AOCS. It is not an attempt to criticize or justify current methods of operation but to present studies and analysis of those operations that have led to the concept of the AOCS.

1.1 Current Procedures

General requirements for a Flight Termination System or FST are provided in the Eastern and Western Range Regulation known as EWR 127-1 *Range Safety Requirements* (2:sec 4-7 to 4-16). This regulation outlines the major objectives of an FST and how they should be accomplished. These criteria must be adhered to at all times when developing any proposed FST.

An overview of what is involved with range safety operations is appropriate at this time. Currently, a ground-based Command Destruct system is employed for all launch operations. This involves tracking of launch vehicles through Range C-band radars. The radar data, or Metric as it has been known historically, are provided to the Mission Flight Control Officers (MFCOs) who are directly responsible to the Range Directors for implementation and execution of actions required to comply with applicable public laws and Department of Defense directives (2). MFCOs have the responsibility of monitoring the progress of a launch vehicle and serve as the sole authority for determining if a launch vehicle should be allowed to continue flight or be terminated. Although the MFCO has some latitude when making decisions concerning the destruction of a launch vehicle, clear guidance is provided in EWR 127-1.

As addressed in EWR 127-1 (2:sec 7.3), there are several situations that would normally result in flight termination action by the MFCO:

a. Valid data show the launch vehicle has violated established flight safety criteria.

b. The performance of the vehicle is obviously erratic and the potential exists for the MFCO to lose positive control. Termination action may be taken even though the vehicle has not violated flight safety criteria.

c. The performance of the vehicle is unknown and the capability exists to violate flight safety criteria.

As part of the Range Standardization and Automation (RSA) effort, a study was conducted to report the feasibility of using a standardized range to vehicle interface that would be used by both the ER and WR. In that study, the following observations were made concerning methods employed during current range operations:

> Current LRS Range Safety methods utilize a continuous UHF command uplink of analog tones, interfaced to a vehicle equipped with dual command receivers connected to UHF antennas. This methodology relies on an antiquated UHF system comprised of redundant components on the Range and on the vehicle. This Range Safety practice requires dual, independent, off-board systems, a single point-of-failure concept, and a man-in-the-loop to monitor control flights. This practice has considerable impact on the vehicle configuration, mission support configuration, Range equipment siting and the Range/Vehicle interface (3:4)

This study continues to describe problems with codes used to activate commands onboard the launch vehicle and how the codes are not standardized to a single code for all vehicles. It is suggested that three significant problems exist with the current method used during range operations. One, a host of redundant equipment using antiquated UHF techniques is needed. This is a costly method in both maintenance and equipment. Two, the flight vehicle antenna composition is very complex. And third, there is no standard

code being used for the command process on all vehicles. These points are relevant when considering whether to design a new command destruct system or stay with the current system. Conclusions are made that new standards for Range-required real-time telemetry data, metric data and command destruct functions are feasible, desirable and recommended. It is further suggested that an AOCS is a candidate to accomplish this mission and could replace the current UHF Command Destruct system by 2002.

1.2 What is the AOCS?

The AOCS concept was developed to meet the need for a standardized Command Destruct process. This proposed solution is very controversial and represents a large departure from the standard process previously described. The proposed design of the AOCS would use onboard microprocessors which contain the programmed vehicle flight profile and rocket performance parameters with permissible variances. The function of the AOCS would be to monitor vehicle performance and control the destruct sequence of a launch vehicle system. It is the destruct sequence or more specifically the flight termination logic of the AOCS that is the thrust of this research effort.

The AOCS could be developed to incorporate three data inputs consisting of Inertial Guidance (IG) data, Global Positioning System (GPS) data, and Inertial Measurement Unit (IMU) information. The onboard GPS receiver would provide position, velocity and time data to the AOCS. Roll, pitch and yaw information would be obtained from the IG or IMU systems. There would also be an optional "man-in-the-loop" system provided by

the C-band radar that would allow the Flight Control Officer (FCO) to send a "safe" code until errant vehicle behavior breaks the link. When the AOCS logic determines launch vehicle data from two of the three onboard systems has exceeded established pre-launch safety parameters and the optional Range/Vehicle "safe" link is broken, the vehicle automatically destructs.

Potential benefits of the AOCS include:

1. There would be large cost savings through a major reduction of ground equipment such as radar tracking systems and data relay devices.
2. GPS data provides an accuracy capability superior to ground-based radar.
3. Optional removal of "man-in-the-loop" procedures from flight termination decisions would reduce the number of personnel needed to perform a launch.

1.3 Approach of this research effort

As stated previously, the main thrust of this research effort is the development of the AOCS flight termination logic. The first step in accomplishing this task was generation of launch vehicle information that could accurately simulate data obtained from a GPS receiver on a launch vehicle. Use of the trajectory simulation program BOOST was invaluable in obtaining this data. The Peacekeeper ICBM was the launch vehicle chosen for simulation.

There are several reasons the Peacekeeper was used as a model for this research. One, information from a 1990 test launch involving a GPS receiver onboard a Peacekeeper was

easily obtained. This information was used as a guideline to format data developed using BOOST. Two, a BOOST constant thrust model for the Peacekeeper (see appendix C) could generate simulated data consistent with real data. And three, data from previous Peacekeeper launches are readily available and can be used to verify the accuracy of the simulated data that are obtained through the BOOST program.

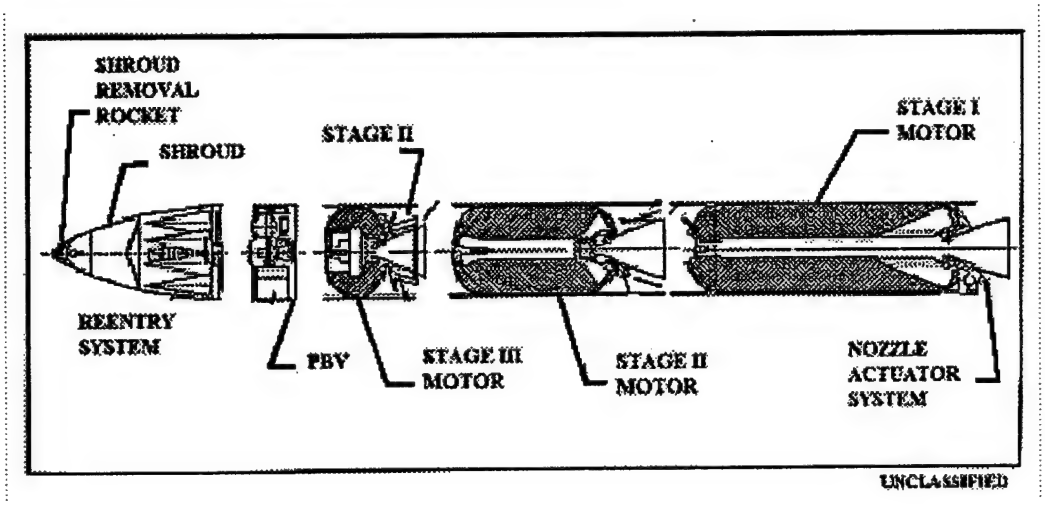

Figure 1 Peacekeeper ICBM Diagram (18)

Figure 1 shows a detailed drawing of the Peacekeeper ICBM. The specific length and thrust of each stage are listed in Table 1. Lengths are given in meters, thrusts are in Newtons. The overall diameter is 2.35 meters. The thrust for Stage 4 and the Payload are not used during this research and are therefor not included in the table.

Table 1

LENGTH AND THRUST PARAMETERS FOR THE PEACEKEEPER ICBM

	Stage 1	Stage 2	Stage 3	Stage 4	Payload
Length	7.50	5.34	2.33	1.07	4.43
Thrust	2,187,538	1,226,766	304,178	N/A	N/A

Although many options were available for developing the decision logic on the AOCS, the one chosen was the Expert System tool CLIPS. CLIPS is a forward chaining rule-based language that has inferencing and representation capabilities. CLIPS was designed at NASA/Johnson Space Center for the specific purposes of providing high portability, low cost and easy integration with external systems (8:373).

Once simulation flight data are obtained from BOOST and the EXMIS program developed using CLIPS, four separate flight scenarios are evaluated. These evaluations are reviewed with results and recommendations made in the final chapter. The specifics concerning the development of both the BOOST data and EXMIS program are discussed in detail within the following chapters.

II. Related Efforts

2.1 Introduction

For approximately the past decade there has been widespread interest throughout the aerospace community in utilizing developments, both scientific and technical, from the area of Artificial Intelligence (AI). For instance, future NASA and U.S. Air Force activities in space will most likely rely on the effective use of some key AI components. Although AI contains many components, there is one AI component in particular that is of concern to this research effort: Expert Systems.

Expert Systems is a unique branch of AI that makes extensive use of specialized knowledge to solve problems approximating the level of a human **expert**. An expert is a person who has expertise in a certain area. That is, the expert has knowledge or special skills that are not known or available to most people (8:2). Expert systems, like human experts, are designed to be experts in one domain area. An expert system designed to play chess would not have knowledge about medicine just as a human chess expert would not be expected to have expertise in medicine. Expertise in one problem domain does not automatically carry over to another (8:3).

A source of invaluable information on the subject of expert systems and their uses in the aerospace design environment is the proceedings of the annual Goddard Conference on Space Applications of Artificial Intelligence. The following sections discuss research

projects presented at this conference during various years. These projects were conducted in the area of expert systems with the intent of using them in aerospace applications. Although these projects do not correlate exactly with what is accomplished during this research effort, they do give a good background as to what has been attempted in the use of expert systems for space applications.

2.2 MOORE and REDEX: Two Diagnostic Expert System

The use of expert systems as a diagnostic tool was one of the first areas where this relatively new technology was found to be very useful. The two expert systems, MOORE and REDEX, although designed for different space systems, perform similar tasks. Both systems assist in diagnosing problems with their specific space systems. Each is discussed in this section starting with the MOORE system.

2.2.1 The MOORE Expert System (11)

A general description of MOORE shows it as a rule-based prototype expert system that assists in diagnosing operational Tracking and Data Relay Satellite (TDRS) problems. This system assists spacecraft engineers at the TDRS ground terminal in troubleshooting problems that are not readily solved with routine procedures, and without expert counsel (11). The prototype system is used to diagnose antenna pointing problems that may occur within the Attitude Control System (ACS). This is a critical area of concern with any

satellite system but is of particular importance to a tracking and data relay system where accurate antenna pointing is crucial.

MOORE was designed by a team at the NASA/Goddard Space Flight Center and was introduced as a proof of concept project in October 1986. It was foreseen as a system that could reduce problem solving time, increase mission support time, and provide an ideal operator training tool. The application of an expert system appeared to be more feasible and beneficial to some of the TDRS subsystems than to others and the choice of the ACS was made based upon several determining factors. The ACS is complex and warrants the development of an expert system, and there is a history of well established and understood ACS problems that could be used to develop an expert system.

The development process for MOORE can be broken down into several separate steps. First, the problem domain of the system was chosen. As stated before, the ACS was selected to fill this role. Then a qualified expert in the ACS area of the TDRS satellite was needed for the purpose of obtaining a knowledge base from which to develop an expert system. Mr. Robert J. Moore of TRW was involved with the design, assembly, integration, test and operations of the ACS for TDRS and was therefore a logical choice. Next, the researchers were faced with choosing a development tool that could meet the criteria of cost, ease of assimilation, speed, size, and vendor support. The Personal Consultant Plus (PC Plus) by Texas Instruments was chosen.

One of the main difficulties and time consuming processes involved with developing an expert system is obtaining a knowledge base from the expert. This project was no different as the researchers found difficulty in obtaining information due to the expert's limited available time. Once the initial knowledge acquisition interviews were conducted, a skeletal structure of the entire anomaly resolution process became apparent. The research team determined that a domain consisting of the entire TDRS ACS was too comprehensive for a prototype expert system. The team decided to narrow the domain to a manageable subset of the ACS. The natural heuristic leaps that an expert would make when dealing with a familiar anomaly were investigated in an effort to capture the expert's thought process when dealing with such problems.

After obtaining the proper knowledge base for the ACS subset, the MOORE expert system was constructed. MOORE operates by querying the user for the spacecraft problem symptoms, then collecting and analyzing this data and, via forward chaining rules, leaps of inference are made that provide intermediate conclusions (11). The successful development of this prototype expert system proved that a diagnostic expert system was feasible and had beneficial application to the TDRS program.

2.2.2 The REDEX Expert System (12)

One of NASA's big challenges for the 1990's is to contain operation and maintenance costs for its mission operations. This challenge has prompted many NASA organizations to evaluate expert systems technology as a means of capturing the specialized expertise of

operations and maintenance personnel. REDEX is one answer to this challenge. REDEX is an advanced prototype expert system that diagnoses hardware failure in the Ranging Equipment (RE) at NASA's Ground Network (GN) tracking stations (12). The REDEX system is designed to reduce troubleshooting time by helping the RE technician identify faulty circuit cards or modules that need replacement. One of its main features is a graphical user interface that uses color block diagrams and layouts to illustrate fault locations.

The problem NASA was having involved the GN tracking stations that track the Space Shuttle launches and various unmanned spacecraft. The RE is a principal equipment item in the S-Band ranging system at the tracking stations. The RE is designed with a Ranging Internal Monitor (RIM) with 74 test points that are distributed throughout the RE. A fault is indicated when one or more of these points are red. The RE technician uses the RIM point status to begin troubleshooting procedures. REDEX was designed to assist the RE in the troubleshooting process by diagnosing the situation and identifying the faulty circuit cards.

REDEX was developed using an overall framework involving a requirements definition, system design, implementation, and testing. The key component of the REDEX development approach is iteration. The requirements were first informally defined, then prototyped, and then formally documented (12). The design process followed the same general sequence. This approach allowed the researchers to evaluate and validate the re-

quirements and design through prototyping prior to formal documentation. The researchers allowed the requirements and design to evolve throughout the prototyping sequence thus ending with the best system available.

The knowledge base needed to develop REDEX was acquired through three main sources: technical documentation (a study of design documents, circuit diagrams, operations manuals, etc.), training classes held by NASA to train new operators and technicians how to operate, troubleshoot, and repair the RE, and interviews with experts in troubleshooting the RE. Once this base was achieved, the knowledge required to diagnose RE faults was represented using rules in an IF THEN format.

REDEX was designed using five evolutionary prototype stages. The objective of the first stage was to demonstrate the feasibility of representing one functional subset of the RE through RE troubleshooting rules using the Prolog language. The second stage objective was to encode the knowledge to diagnose faults in the entire RE network. The third stage had the objective of developing and evaluating the concepts of the user interface design. The fourth stage fully implemented the user interface. The fifth and final stage corrected diagnostic errors discovered during testing and added a capability to communicate with the RE.

2.3 ISTAR: A More Recent Application (16)

The development of the Intelligent System for Telemetry Analysis in Real-time (ISTAR) was initiated in May of 1988 by the Air Force Space and Missile Systems Center (SMC), Inertial Upper Stage (IUS) Office. This system was designed as an advanced vehicle monitoring environment incorporating expert systems and analysis tools. Over a five year period the system progressed from rapid prototype to operational system (16). The prime developer for this project was the Aerospace Corporation, a federally funded research and development contractor.

At the time of ISTAR's development, several trends were taking place in the world of defense spacecraft that were making the mission support task increasingly difficult. First, defense spacecraft were becoming more and more complex. The massive amounts of data received by ground personnel must be quickly and reliably interpreted through slow and sometimes unreliable manual means. Second, the number of defense space vehicles and booster launches that need support were increasing. Replacement satellites for existing constellations and new satellite systems will place an even heavier burden upon mission support operations in the future. Third, the number of qualified mission controllers were diminishing because of retirements. Typically, the controllers with the greatest amount of experience and expertise are the ones closest to retirement (16:329). Finally, the trend towards downsizing and defense cutbacks have taken their toll on the number of qualified personnel to accomplish the mission.

2.3.1 The ISTAR System

ISTAR was foreseen as a solution to a majority of these problems. The system was first designed to be operational with the Spacecraft Monitoring And Real-time Telemetry (SMART) system within Mission Control Center #8 of the Air Force Consolidated Space Test Center (CSTC). SMART is a real-time data acquisition and analysis system based on the System-90 Product by Loral Data Systems (16:332). ISTAR is designed to run in the background on existing workstations within the SMART system in either real-time or playback mode. In *real-time* mode, the Expert System Process analyzes dynamic telemetry data and detects and diagnoses anomalous conditions. If an anomaly is detected, ISTAR is brought to the foreground, allowing the controller to investigate the problem further.

Several Knowledge bases can be invoked by the user through the ISTAR system. A Status Knowledge base can be used to provide general vehicle status information to the user through certain status messages. One such message could alert the mission controller to vehicle events such as significant changes or limit violations of voltages or currents within a subsystem of the satellite. Another Knowledge base can monitor specific phases of the mission such as deployment or solid rocket motor burns. Knowledge bases within ISTAR monitor specific subsystems or components of the IUS. Such subsystems as the Power Distribution Unit and Signal Conditioner Unit Multiplexer are monitored through

the entire mission. The Knowledge base is configured to detect anomalies that are most likely to occur, or most critical to detect.

2.3.2 Testing ISTAR

Prior to its operational use, ISTAR went through extensive testing at CSTC to insure that it would reliably operate on current SMART workstations, yield results consistent with SMART, and function without adversely affecting the operation or performance of the SMART network (16:335). A groundrule was established to insure the system would function within these parameters. Simply stated, no expert system results would be used without verification from an official Air Force telemetry data system. After several simulations were completed, it became apparent that the ISTAR system would provide valuable and timely information to the mission support team. This performance generated support from SMC and Aerospace Corporation for its continued development and use.

2.3.3 Lessons Learned

ISTAR represented the first significant attempt to acquire an expert system capability for the Air Force SMC Space Launch Operations Program. Over the five years of development and operation many lessons were learned (16:336). Some of the most significant ones relating to this research effort follow.

Start with simple, rapid prototypes, and proof of concepts - By building a prototype and showing the solution of a very limited subset of the entire problem, a new perspective on program needs and problems to be solved will arise

Use Commercial Off The Shelf (COTS) products whenever possible - Support costs are lower, products are more adaptable, and COTS products are thoroughly tested.

Start with simple, well-defined knowledge bases first - Start simple and enhance the knowledge base through an evolutionary process.

Tackle problems that are best suited for expert systems- Not every problem is suitable for an expert system. Detecting combinations of vehicle events that indicate a problem is an appropriate use of an expert system.

Deal with real-time data and control issues early - A technical challenge is determining how to connect to and process a real-time data stream. It is important to demonstrate the capabilities of the system in the operational environment early in order to increase the credibility of the system.

2.4 Autonomous Power Expert System (21)

A good example of an expert system used in a space system application is the Autonomous Power Expert (APEX) system. The APEX system was developed by the Space Electronics Division at the NASA Lewis Research Center. It was designed to monitor and diagnose fault conditions within the Space Station Freedom Electrical Power System (SSF/EPS) through the use of a rule-based expert system.

Design of the expert system is based on a model that consists of objects organized into frames which are combined to represent an integration of object-oriented programming and frame-based knowledge representation (21:148). Through the use of forward and backward chaining rules, expert reasoning is emulated and provides for fault detection and isolation. A load profile is provided from a remote power scheduler program that contains information about expected values and operating conditions for each load.

The research team concluded that an expert system can check more test points, more often than a human operator can, and do so without fatigue (21:156). They also found that having expert knowledge continuously available to monitor and diagnose faults would provide a great advantage over the current human methods. These were seen as valuable benefits for a system such as a space station that requires continuous feedback and autonomous control. It is foreseen that much of the burden that is currently placed upon human operators can be relieved by expert systems such as APEX.

2.5 CLIPS as a Knowledge Based Language (10)

This study compares the CLIPS language with three other AI languages with regard to the processing they provide for the implementation of Knowledge Based Systems (KBS). The three other languages are LISP (*LIS*t *P*rocessing language), Prolog (*Pro*gramming in *L*ogic), and OPS5 (*O*fficial *P*roduction *S*ystem, version 5). These languages were all developed to enhance the building of KBSs by providing a direct method of encoding both data and procedural knowledge (10:33). The researchers determined that the most common language used for developing AI applications was LISP. CLIPS was the most recently developed language of the set studied by the research group.

The researchers discussed the differences between a KBS language and a conventional language such as Pascal, C, or FORTRAN. They describe a KBS language as one based on a set of rules that act like functions in a conventional language. The rules are triggered by data rather than program flow. This study determined that if the order of decisions can

be predetermined and remains constant regardless of the data, then a conventional language is a more appropriate selection. Another important difference is the way variables are handled. In the AI languages the variable only has meaning within the particular rule in which it is located. There are no global variables as in FORTRAN or Pascal.

The study finds several advantages to using CLIPS over the other four languages. For instance, CLIPS is the only language of the set that was designed especially to be embedded within another system. CLIPS also provides language constructs to perform algorithmic tasks. Another feature that is useful in CLIPS is the ability to assign weights (called salience values) to rules. The rule with the highest salience value is the rule that fires next (10:38).

III. Missile Trajectory Model

3.1 Introduction

During the formulation of this research project, it was discovered that *real* GPS data obtained during tracking of actual launch vehicles during flight were unavailable in the quantities necessary to verify operation of the EXMIS program. Therefore, a different means of obtaining missile trajectory information was needed. Computer simulation was the desired method since many different scenarios would be possible and alteration of the information could be performed easily. Using real GPS tracking data as a standard, several options were investigated as to their utility of providing accurate simulated GPS and inertial data.

The development of a missile simulation using Matlab® (14) was pursued but the model developed could not provide a good approximation of the dynamics experienced by a missile flying in 3-D space. The Matlab program was therefore dropped from consideration. However, a program called BOOST was found to be more user friendly and capable of generating the types of data necessary to test EXMIS accurately. BOOST is used by the Ballistic and Space Design Branch (TANB) at the National Air Intelligence Center (NAIC) and was easily accessible for use during this research project. BOOST is explained in more detail during the following sections.

3.2 Defining the Equations of Motion

A dynamics problem, such as the one investigated during this research effort, can not be solved without first developing the equations of motion (EOM). Thorough and extensive use of "BOOST: A General Six Degree of Freedom Powered Flight Trajectory Simulation Computer Program" by Anderson (1) was made in order to accomplish this task. This source gives great insight into the dynamics of a missile during flight and how the BOOST program computes and develops certain equations.

3.2.1 Coordinate Systems

It is appropriate to describe the various coordinate systems used during this study. The BOOST program has four basic coordinate systems available. Three are Earth-oriented systems and the fourth is missile-oriented. The first is an Earth-centered system that is fixed at the time of launch. The X and Z axes are in the equatorial plane. The X axis passes through zero longitude where the Y axis is coincident with the polar axis and the Z axis is perpendicular to the X-Y plane.

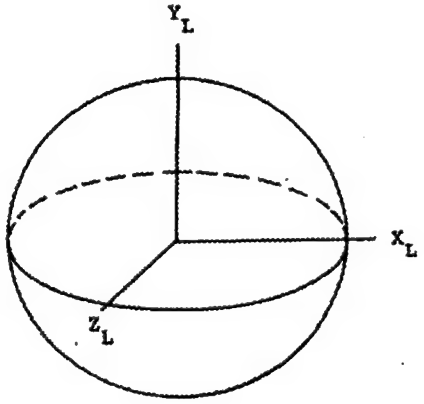

Figure 2 Earth-Centered Inertial System (1)

The second system is an earth-centered system that rotates with the earth and is coincident with the inertial system at the time of launch.

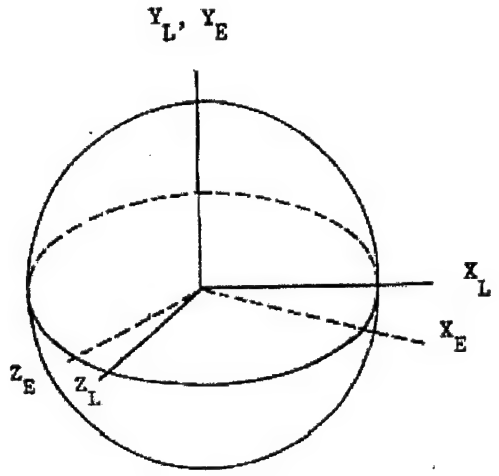

Figure 3 Earth-Centered Rotating System (1)

The origin for the last earth-oriented system is located by an input latitude and longitude and the orientation is controlled by an input azimuth. This allows the user to specify the exact launch location. The launch location specified for this study is Vandenberg AFB with Latitude = 34.7 ^{0}, Longitude = 120.6 ^{0}, and an azimuth of 200 ^{0}. The coordinate system remains fixed with respect to the Earth during the entire missile flight.

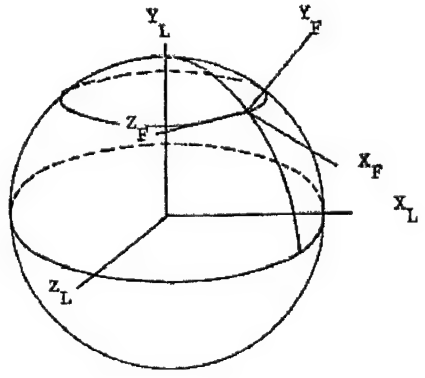

Figure 4 Launch-Centered System (1)

The missile coordinate system is centered at the missile center of mass. The 1 axis represents the roll axis, the 2 represents the pitch axis, and the 3 represents the yaw axis. Rates of rotation are also determined about each of these axes.

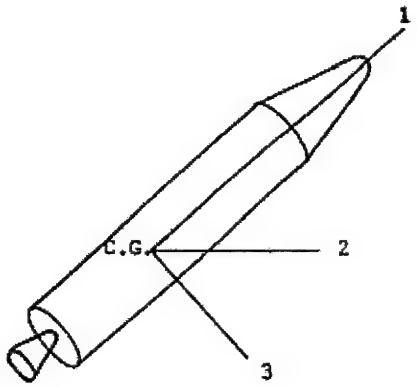

Figure 5 Missile-Coordinate System(1)

3.2.2 Initial Conditions

BOOST uses a numerical integration scheme to obtain inertial positions and velocities. This scheme involves a Predictor-Corrector method wherein velocities and positions are first predicted for the next point in the trajectory based on previous and current values of acceleration, velocity and position (1). From the predicted velocities and positions, BOOST is able to calculate the forces acting on the missile and determine its acceleration. Certain initial equations must be computed to begin the process. These equations involve the inertial position and velocity of the missile at the launch point and are used to develop the equations of motion. The Predictor-Corrector method used by BOOST will be described later in this chapter. Following are the initial equations that lead to the EOM.

Inertial Position

$$r = R_E + h_o$$
$$X_{LO} = X_L = r\cos\phi_{GO}\cos\lambda_o$$
$$Y_{LO} = Y_L = r\sin\phi_{GO}$$
$$Z_{LO} = Z_L = r\cos\phi_{GO}\sin\lambda_o$$

Inertial Velocity

$$\dot{X}_{LO} = \omega \cdot Z_{LO}$$
$$\dot{Y}_{LO} = 0$$
$$\dot{Z}_{LO} = -\omega \cdot X_{LO}$$

Where the following list defines the variables used. (NOTE: All values used by BOOST are in the English measuring standard but outputs are converted to SI units):

r = Radius to the missile

R_E = Radius of the Earth

h_o = Launch altitude

X_{LO}, Y_{LO}, Z_{LO} = Inertial coordinates of launch point

ϕ_{GO} = Geodetic launch latitude

λ_o = Launch longitude

ω = Angular rate of Earth rotation

The direction cosines for the missile orientation matrix that transforms missile coordinates into earth-centered coordinates are also provided as a 3 X 3 matrix **R** below (1:7.1).

$$R_{11} = \cos\lambda_o \cos\phi_{GO}$$
$$R_{12} = \sin\phi_{GO}$$
$$R_{13} = \sin\lambda_o \cos\lambda_{GO}$$
$$R_{21} = \cos\lambda_o \sin\phi_{GO} \sin\beta_o + \sin\lambda_o \cos\beta_o$$
$$R_{22} = -\cos\phi_{GO} \sin\beta_o$$
$$R_{23} = \sin\lambda_o \sin\phi_{GO} \sin\beta_o - \cos\lambda_o \cos\beta_o$$
$$R_{31} = -\cos\lambda_o \sin\phi_{GO} \cos\beta_o + \sin\lambda_o \sin\beta_o$$
$$R_{32} = \cos\phi_{GO} \cos\beta_o$$
$$R_{33} = -\sin\lambda_o \sin\phi_{GO} \cos\beta_o - \cos\lambda_o \sin\beta_o$$

The angles are defined below.

ϕ_{GO} = Geodetic launch latitude

λ_o = Launch longitude

β_o = Launch azimuth

3.2.3 Equations of Motion

From the given initial conditions and through the use of the previously described equations, the equations of motion can now be developed. A simple representation of the Equations Of Motion (EOM) are given here. For a more rigorous derivation of all parameters used by the BOOST program the reader should refer to the BOOST manual (1). Using Newton's fundamental equation, F = ma, the inertial components of acceleration are solved. It is assumed there are only four forces acting on the missile at any given time: the axial thrust, an axial aerodynamic force, the normal force, and the force due to gravity. Each force will first be defined and then used to find the EOM.

The axial thrust is found by first finding the total thrust T, which BOOST finds using a user input parameter known as the specific impulse or I_{SP}. Thus, the general equation for T is as follows (18:24):

$$T = I_{SP}\dot{m} / g_o$$

$\dot{m}$ = mass flow rate

g_o = *acceleration of gravity* 32.17 $^{ft}/_{sec}$ *or* 9.8 $^{m}/_{sec}$

The axial thrust is then computed using input dimensions specified for the particular launch vehicle selected by the user. These dimensions are input during the setup phase by a model subprogram. The subprogram used for this research project is the Peacekeeper constant thrust model (see Appendix C).

The equation for axial thrust is (1:2.1):

$$T_{AX} = \sqrt{T^2 - \left[\left(\frac{L_g - L_p}{L_i - L_g}\right)^2 \cdot \left(N_2^2 + N_3^2\right)\right]}$$

T = total thrust

L_g = distance from nose to center of gravity

L_p = distance from nose to center of pressure

L_i = length of missile

N_2, N_3 = aerodynamic forces along missile 2 and 3 axes

Using the axial thrust, the acceleration for each direction can be obtained (1:2.1).

$$\ddot{X}_L = \tfrac{1}{M}\left[\left(T_{AX} R_{11}\right) + N_X - A_X + W_X\right]$$
$$\ddot{Y}_L = \tfrac{1}{M}\left[\left(T_{AX} R_{12}\right) + N_Y - A_Y + W_Y\right]$$
$$\ddot{Z}_L = \tfrac{1}{M}\left[\left(T_{AX} R_{13}\right) + N_Z - A_Z + W_Z\right]$$

The individual equations, used to define the equations of motion, follow.

$$N = C_{N\alpha}\, q\, S$$

$$A_1 = C_A\, q\, S \qquad A_2 = N_2 = \frac{V_2\, N}{\sqrt{V_2^2 + V_3^2}} \qquad A_3 = N_3 = \frac{V_3\, N}{\sqrt{V_2^2 + V_2\, N}}$$

$$A_X = A_1\, R_{11} \qquad A_Y = A_1\, R_{12} \qquad A_Z = A_1\, R_{13}$$

$$N_X = \left(N_2\, R_{21} + N_3\, R_{31}\right)\left(1 + \frac{L_g - L_p}{L_i - L_g}\right)$$

$$N_Y = \left(N_2\, R_{22} + N_3\, R_{32}\right)\left(1 + \frac{L_g - L_p}{L_i - L_g}\right)$$

$$N_Z = \left(N_2\, R_{23} + N_3\, R_{33}\right)\left(1 + \frac{L_g - L_p}{L_i - L_g}\right)$$

$$W_X = -M\,\mu\frac{X_L}{r^3} \qquad W_Y = -M\,\mu\frac{Y_L}{r^3} \qquad W_Z = -M\,\mu\frac{Z_L}{r^3}$$

The following symbols are defined for clarification.

M = Mass of the missile (an input value)

N = Total aerodynamic forces

N_X, N_Y, N_Z = Aerodynamic forces along the X, Y, Z axes

$N_2, N_3,$ = Aerodynamic forces along the missile 2 and 3 axes

q = Dynamic pressure

A_X, A_Y, A_Z = drag forces along the X, Y, and Z axes

A_1, A_2, A_3 = drag forces along the missile 1, 2, 3 axes

C_A = A function of Mach number

$C_{N\alpha}$ = Also a function of Mach number

W_X, W_Y, W_Z = Components of gravity forces in X, Y, Z directions

V_2, V_3 = Components of relative wind in the missile 2 and 3 directions

X_L, Y_L, Z_L = Inertial coordinates of missile position

S = The Missile surface area

Some values used within the BOOST program are obtained through lookup tables that have been developed using historical data from previous launches. An example is the values for C_A.

3.3 BOOST: A Trajectory Simulation Program

The BOOST program was selected for this research project because of its ease of use and history of excellent performance. It is currently being used by the TANB Section of the National Air Intelligence Center (NAIC) at Wright-Patterson Air Force Base, Ohio.

This afforded an opportunity to work with individuals who were knowledgeable about BOOST and had a wealth of documentation available on many different launch vehicles.

3.3.1 Description of BOOST

BOOST is described as a general six degree of freedom trajectory simulation program capable of simulating a multi-staged launch vehicle (1). It was developed to provide a preliminary design analysis and quick reaction capability for performance analysis of existing launch vehicles. The means available for controlling the vehicle attitude include a zero lift method, predetermined pitch and yaw programs, inertial gravity turn, and Appazov angle of attack steering. Only the pitch and yaw programs were used for attitude control during this research effort.

The zero lift method involves calculation of a pitch rate and, if called for by the program, a yaw rate that causes the missile to trim into the wind. This maneuver effectively aligns the missile 1 axis with the relative wind velocity vector resulting in a zero total normal force, hence no lift force. BOOST treats attitude rates independently making it possible to fly a missile with zero lift in the pitch plane while changing the yaw rate through use of the yaw rate control law. The pitch and yaw rate programs consist of specifying missile rotations about the missile 2 and 3 axes, respectively, in the form of an input rate versus time. The program applies the rate from a lookup table, determined by

the user, which corresponds to the current time. It is therefore possible to turn (that is yaw) or adjust the pitch of a missile at a predetermined time during the flight program.

All inputs to BOOST are made through variables found in an input namelist. The user must first find the correct input symbol for the type of function that BOOST is required to perform. This is accomplished with the help of an appendix in the BOOST manual (1). This appendix covers all input variables or symbols that are available to the user. Once the user has found the correct input variable, it must be input into BOOST through a setup model. A preprogrammed setup model was available and used for this research project and is discussed in a later section. For a listing of input variables used during this research effort, the reader should consult Appendix A.

In order to specify completely the nature of the missile whose trajectory is to be computed, it is necessary for the user to input all pertinent data about the missile (1). This includes the missile configuration (thrust at each stage, weights, lengths, etc.), launch point coordinates, missile control data (attitude rates, launch azimuth, etc.) and miscellaneous values such as drag curves. Once this general information is provided to the program, slight variations in certain variables can result in completely different flight scenarios.

BOOST calculates the path of the missile by solving the equations of motion in an inertial coordinate frame. From the given inputs BOOST computes the trajectory of the missile in a step by step method by resolving all of the forces acting on the missile, at any

given time, into accelerations. The accelerations are integrated to give velocities, which in turn are integrated to give positions. The missile orientation is calculated in the same manner by integrating the rotations about each of the missile axes and relating the positions of the axes to the inertial frame. Through this step by step process, BOOST calculates the desired missile trajectory from launch to impact.

As mentioned previously, the integration scheme used by BOOST to obtain the inertial positions and velocities is a predictor-corrector method. First, velocities and positions are predicted for the next point in the trajectory based on prior values of acceleration, velocity and position. From the predicted values, BOOST calculates the forces acting on the missile and determines the accelerations for each direction. With the new accelerations, a velocity and position corrector equation is developed and applied. If the predicted and corrected values agree within a specified tolerance, the values are accepted and used as a point on the trajectory. If the values do not agree, the corrected velocities and positions are used to obtain accelerations and then recorrected. The program allows a maximum of 10 recorrections. If the scheme fails to agree after the 10 iterations, trajectory computations are halted.

A general example of the equations used in the Predictor-Corrector method are given in the BOOST manual (1:3.1) and are shown here for illustration. The equations for the inertial X, Y, and Z coordinates are identical and therefore only the X coordinate equations

will be shown. The subscripts n-1, n and n+1 refer to time steps in the trajectory computation with n-1 being the previous time step, n the current one, and n+1 the next (1:3.1).

$$\dot{X}^{P}_{n+1} = \dot{X}_n + \Delta t\, \ddot{X}_n \qquad \text{Discontinuous Predictor}$$

$$\dot{X}^{P}_{n+1} = \dot{X}_{n-1} + 2\Delta t\, \ddot{X}_n \qquad \text{Continuous Predictor}$$

$$X^{P}_{n+1} = X_n + \frac{\Delta t}{2}\left(\dot{X}^{P}_{n+1} + \dot{X}_n\right) \qquad \text{Position Predictor}$$

$$\ddot{X}^{P}_{n+1} = f\left(X^{P}_{n+1}, \dot{X}^{P}_{n+1}\right) \qquad \text{Equations of Motion}$$

$$\dot{X}^{C}_{n+1} = \dot{X}_n + \frac{\Delta t}{2}\left(\ddot{X}^{P}_{n+1} + \ddot{X}_n\right) \qquad \text{Velocity Corrector}$$

$$X^{C}_{n+1} = X_n + \frac{\Delta t}{2}\left(\dot{X}^{C}_{n+1} + \dot{X}_n\right) \qquad \text{Position Corrector}$$

3.3.2 Peacekeeper Constant Thrust Model

The Peacekeeper constant thrust model was designed by Mr. Dale B. Witt of the FASTC/TANB office at NAIC in June 1993. The model is normally a classified subprogram for the BOOST trajectory simulation program, but, after eliminating certain pieces of information it was unclassified for use during this research. The model supplies the input information needed by BOOST to begin trajectory simulation for the Peacekeeper ICBM. By using this model, consistency was maintained since certain information utilized for this research effort was specific to the Peacekeeper. For example, the GPS information, adopted as a guide for input data to EXMIS, was obtained from a Peacekeeper launch.

The complete structure of the Peacekeeper model can be found in Appendix C. A short breakdown of the model is given here for clarification. The Post Boost Vehicle (PBV) and Reentry Vehicle (RV) are removed to make the model unclassified. The first few lines give a background concerning the initial conditions of the simulated launch and are for information purposes only. The launch height is specified as 0 ft and the temperature is 72° F.

The Peacekeeper is launched using a cold launch system that propels the missile out of the launch tube. Vehicle motor ignition takes place approximately 2.7 seconds after cold launch. The thrust for each stage is input using the T1(I) variable where I equals the stage number. The total weight of the launch vehicle, including payload, is 195,444 lbs or 88,652 kg and is put into BOOST through the W1(1) variable. Time of flight, weights at each stage, and weight decay rates are controlled through the TF(I), W1(I) and WD(I) variables. The AC(I) variable allows the user to designate the thrust gradient for each stage.

The flight scenarios studied during this research project were accomplished by changing certain input variables at specified times during the simulation. The PSICV variable is designed to control the missile yaw rate. The variable consists of two separate inputs: time, in seconds, that specifies the start of the yaw rate, and the rate of yaw in deg/sec. The THACV variable accomplishes the same function for the missile pitch rate. These two variables allowed the program to simulate missile tumbling about an axis resulting in

an unstable launch vehicle profile. This simulation was vital for testing the capability of the EXMIS program to detect accurately problems with the launch vehicle and then perform appropriate actions. The SDELT variable allows the user to specify the interval at which trajectory data is computed and output from the program. The interval used for all simulations was 1/10 sec. The last variable to be described is the PRTBL variable. This variable defines the interval at which the outputs from the program are printed. The frequency specified was 1/2 sec. This created a good data history for the simulated flight and provided substantial information in which to test the EXMIS program.

3.4 Summary

The BOOST program was invaluable to this research project. The ease of use and documentation support made it the perfect tool with which to obtain the trajectory information needed for testing the EXMIS program. The input variables, described in this chapter, allow the user to customize the trajectory program as needed. This makes the program versatile and viable for many trajectory simulation projects. Therefore, this program is recommended to any researcher looking for a trajectory simulation program with these particular assets.

I V. The Expert System

4.1 Introduction

Expert systems is a branch of AI that uses specialized knowledge to solve problems at the level of a human expert. An expert is defined as a person who has expertise or knowledge in a certain area that is not known to most people. Therefore, an expert system can be defined as a computer system which emulates the decision-making ability of a human expert (8:1). Usually, an expert's knowledge is specific to one particular area or problem domain as opposed to knowledge about general problem-solving techniques. Expert systems are similar in this respect since they are generally designed for use in one problem domain.

4.1.1 Design of an Expert System

The development of an expert system is a time consuming and often rigorous undertaking. The process has several stages that must be followed in order to create an effective and efficient system. The stages outlined in the next few paragraphs are a general guide for developing an expert system. The specific process used during this research project will be described later in the chapter. The first step in the process is to define the task or problem and decide whether an expert system would be appropriate for solving the problem. Tables 2 and 3 and give some characteristics of tasks suitable and not suitable for expert system development.

Table 2

SUITABLE TASK CHARACTERISTICS (13:130)

- The proposed system will save time and money.
- An expert for the problem domain is available.
- Nonexperts require the expertise.
- The task can be fully described with facts, rules, and algorithms.
- The expertise to be modeled is narrowly focused.
- The time required to perform the task is between a few minutes and a few days.
- The task may have a logical complexity such that no single human expert could perform it without help from computers or manuals.

Table 3

NON- SUITABLE TASK CHARACTERISTICS (13:130)

- The task is based on hard-to-define knowledge, common sense, or intuition.
- The actions of the expert may require human skills that are difficult to computerize.
- An expert could not describe in detail, over the phone, how he performs the task.
- The task is performed by most people.
- It is unclear whether the intended users would employ the system.

When the task has been identified as a good candidate for an expert system, the "knowledge engineer" must obtain a knowledge base about the problem. This can be accomplished by eliciting information from one or more experts in the field or from manuals used in performing the task. The knowledge engineer must identify the key concepts and facts about the problem and ways they are linked together to solve the problem. This is often the most time consuming part of the expert system development process. It is often

desirable to define a small piece of the problem to tackle first. This allows the knowledge engineer to test some of the basic problem-solving strategies used by the expert when performing the task.

Once a knowledge base is derived from the available sources, the knowledge must be formalized into a set of rules and a structure built to organize them. A rule is similar to an IF-THEN statement used in a procedural language such as Ada. The rules are used to represent relationships in terms of condition-reaction pairs where IF is the condition part and THEN the action part. The condition part (IF) of the rule is also referred to as the premise, antecedent, or left-hand side (LHS). The action part is also referred to as the conclusion, consequent, or right-hand side (RHS) (13:234). After all rules are developed, they are programmed into an AI language such as Lisp or CLIPS. The knowledge engineer must continually evaluate the rules to ensure they accurately represent the problem.

Now that the rules have been defined and entered into an AI language, they are used to construct a workable prototype. The first-out or "rough draft" version of the prototype is tested and experimented with to determine its utility. The rough draft prototype is refined through an iterative process until it is determined to be acceptable by both the expert and knowledge engineer. The prototype expert system is evaluated by a user in the field. Refinements are made from user suggestions until all individuals involved are satisfied with the prototype. The expert system is finally accepted and used on real cases.

4.2 CLIPS: An Expert System Language

CLIPS was designed at the Johnson Space Center/NASA with the specific purpose of providing a highly portable, low cost expert system language for developing expert systems. In keeping with these goals, NASA has provided CLIPS free of charge to Government users and Government agencies. CLIPS is written in the C programming language giving it speed and portability. Because of its portability, CLIPS has been installed on a wide variety of computers ranging from PC's to CRAY supercomputers (8:374). Some of CLIPS' characteristics are smooth integration with external systems and easy embedability within a main system.

CLIPS is a forward chaining rule-based language. In a rule-based system, the inference engine determines which rule or rules are satisfied. The inference engine corresponds to what is known as a cognitive processor within the human brain. The cognitive processor tries to find the rules that will be activated by the appropriate stimuli (8:13). Similarly, the inference engine in an expert system controls the overall program execution by looking for facts that will activate a particular rule. The forward chaining process involves steps that go from the reasoning of facts on to the conclusions resulting from those facts. For example, if you see that it is raining before you leave home (the fact), then you know you should take an umbrella (the conclusion) (8:28). As mentioned before, the reasoning process is performed by the inference engine.

When designing an expert system, care must be taken to choose a development language that will allow the expert system to perform it's intended task. Analysis of all available languages is one of the first steps in the design of an expert system. An incorrect choice at this stage of development can cause many problems further down the line. Matching the needs of the expert system with the characteristics of the available development languages was a logical procedure for this research project.

When analyzing CLIPS for possible use as the expert system language, it was found that CLIPS is well suited for writing control systems. According to one expert systems researcher, " The CLIPS language is a good choice for writing a control system or simulation of a control system. The rule based nature of the CLIPS language provides an intuitive and quick medium for developing control rules " (10:39). It was determined that the prototype expert system developed during this research project would integrate with many external systems including GPS receivers and inertial measuring equipment. It will also be imbedded within the AOCS system. These factors combined to make CLIPS an excellent choice for developing EXMIS.

4.2.1 Structure of CLIPS

CLIPS consists of three basic components. These three components are: a fact-list, knowledge base, and inference engine. In order to solve a problem, CLIPS must first have information or data with which it can reason. Information is known as a **fact** within CLIPS. A fact consists of one or more fields enclosed by parentheses. An example is:

(day Monday) or (day Tuesday). In this instance, the two facts are explicitly related. The first field of the facts (day), is used to describe the relationship of the following fields (Monday) or (Tuesday). Now the facts are related by the field **day**. Each fact is placed in a fact-list that will be used later in the program.

A set of facts can be entered into the fact-list using the **deffacts** construct. This is particularly useful when facts are known to be true before running a program such as initial knowledge or conditions. The general format for a CLIPS deffacts statement is (8:395):

```
(deffacts <deffacts name> [<optional comment>]
  << facts >>)
```

The keyword deffacts is first, followed by the name of the deffacts statement. An optional comment is available to further describe the facts being asserted. Following the name or optional comment are the facts that will be asserted in the fact-list by the deffacts statement. Shown below is a specific example of a deffacts statement:

```
(deffacts initial-status "Initial conditions for the day"
   (sun up)
   (lights off))
```

The name of the deffacts statement is *initial-status*, with a comment following. There are two facts in this deffact statement, (sun up) and (lights off). The facts are asserted and placed on the fact-list by using the CLIPS **reset** command described later in this section.

In order to accomplish it's assigned task, an expert system must have **rules** to go with it's facts. Following is the general format for a CLIPS rule with the name of each part in bold letters (8:386):

```
(defrule <rule name> [<optional comment>]   ; Rule header
   <<patterns>>   ; Left-Hand Side (LHS) of the rule
=>
   <<actions>>    ; Right-Hand Side (RHS) of the rule
```

In this format the entire rule must be surrounded by parentheses or CLIPS will not acknowledge it as a rule. The rule header consists of three separate parts. The first part is the keyword **defrule,** which indicates to CLIPS that a rule is about to be defined. Following defrule must be the **name** of the rule. If a rule is entered that has a name which is the same as an existing rule, the new rule will replace the old rule. CLIPS provides for an optional **comment** after the rule name that allows the programmer to provide additional information about the rule such as it's purpose.

After the rule header are zero or more **patterns** or conditional elements consisting of one or more fields. CLIPS attempts to match the patterns against facts in the fact-list. If all the patterns of a rule match a fact, the rule is **activated** and put on the **agenda**, the collection of activated rules (8:386). The => symbol that follows the patterns of a rule is known as the **arrow**. The arrow signifies the beginning of the THEN part of an IF-THEN rule described earlier in this chapter. The LHS is the part of the rule just before the arrow and the RHS is just after the arrow. The right hand side consists of **actions** that will exe-

cute when the rule **fires**. The term fires means that CLIPS executes the actions of a rule from the agenda in the same manner that a nerve cell (neuron) transmits a nerve impulse (fires) when the proper stimuli is present (8:387).

Often, computer programmers write their initial code for a program in a format known as pseudocode. Pseudocode is used by CLIPS programmers as a way of writing rules in an understandable English format that can later be converted into a CLIPS format. Following is an example of a pseudocode rule:

```
IF the sun has set
THEN turn on the lights
```

This rule could be entered into CLIPS as:

```
(defrule set-sun-now
  (sun set)
=>
  (assert  (action turn-lights-on)))
```

Notice that the parentheses surrounding the patterns and actions must be balanced or a CLIPS error will occur. In this rule, the pattern is (sun set) and the action asserted by the rule is *turn-lights-on*. If the (sun set) fact is subsequently asserted by another rule or deffacts statement it is placed on the fact-list, the rule *set-sun-now* will fire and the assertion *turn-lights-on* will be made.

When there are multiple rules on the agenda, CLIPS automatically knows which rule to fire. CLIPS orders the rules on the agenda by increasing priority and will fire the rule with the highest priority, or **salience**, first. Normally, the agenda acts as a stack with the most recent activation placed on the agenda firing first. Salience allows the most important rules to stay at the top of the agenda no matter when they were added. CLIPS has a keyword called salience that allows the programmer to specify the priority of a certain rule, thus controlling the execution of the expert system. Salience values can be specified from 10,000 to -10,000. If a rule has no salience value assigned by the programmer, CLIPS assigns a value of zero.

The **reset** command is the key method for starting an expert system in CLIPS. The command removes all activated rules from the agenda and all facts from the fact-list (8:396). The reset command will then assert all of the facts from existing deffacts statements. A CLIPS program will not start running unless there are rules whose LHS are satisfied, therefore, when CLIPS is initially started it automatically defines a deffact statement known as the initial-fact statement. Thus, even though no deffacts statements have been defined, a reset will assert the initial-fact and enable the expert system, written in CLIPS, to operate. Following the reset command is the **run** command. The run command tells CLIPS to start looking for rules on the agenda that have been satisfied and should therefore fire.

The previous paragraphs give a brief outline of the general CLIPS language structure and how it operates. For a more detailed description of the CLIPS environment and its particular commands, the reader is encouraged to consult references 7 and 8. Both sources are written by Joseph Giarrantano and delve into the finer issues of writing an expert system using CLIPS.

4.3 EXMIS: The EXpert MIssile System

The EXMIS program is a prototype expert system designed to perform decision logic tasks within the AOCS. The intent of the AOCS is to decrease the dependence that launch vehicles have upon human decision makers. By incorporating an autonomous system capable of performing trajectory analysis and self destruct determination, the need for human *expertise* is decreased. The developmental nature of the AOCS project makes investigation of new technologies, such as expert systems, a logical step in the process.

4.3.1 Establishing Criteria for the AOCS and EXMIS

The AOCS can take advantage of some new technologies available in today's environment that were not available when the current system was developed. For one, it could receive and process flight data from GPS and IMU sources. These data could be used by EXMIS to ascertain if predetermined safety criteria are violated and, if so, take established actions emulating steps a human expert would take in a similar situation. The steps in-

volved with developing an expert system were outlined earlier in this chapter and every effort was made to follow these steps during this research project.

The first step in developing an expert system is to investigate the problem or task and decide whether an expert system is appropriate for that problem. Through discussions with experts in the task area (4, 5), and following the guidelines of table 2, it was determined this task fit most of the criteria of a suitable candidate for an expert system. The following observations were made:

- The proposed expert system would save time and money by replacing costly radar tracking equipment in favor of using GPS and IMU data.
- An expert is available from which a knowledge base can be developed.
- The task can be fully described with facts, rules, and algorithms and follows an established procedure.
- The expertise to be modeled is narrowly focused and unique to this task.
- The time required to perform the task is within the capabilities of an expert system.
- Computers and regulations are used by more than one individual to perform the task.

4.3.2 Obtaining a Knowledge Base

Once it is established that the task is suitable for an expert system, a knowledge base for the task is developed from interactions between the expert and the knowledge engineer. This step was accomplished through telephone interviews with Mr. Allen Dumont, a Senior MFCO for the 30th Space Wing (30 SW) at Vandenberg, CA., and Mr. Fred Foerst, an Instrumentation Engineer also with the 30 SW (4,5). As a SMFCO, Mr. Dumont monitors the performance of launch vehicles in flight and initiates flight termination when required. He described the basic procedures involved with launching and controlling a

missile and how these could pertain to an AOCS type system. Mr. Foerst has considerable knowledge of the proposed AOCS and provided documentation invaluable to this research project.

Using the main regulation governing range safety requirements, EWR 127-1, an insight into the responsibilities of the MFCO was realized. Section 7.2.3 of this regulation outlines these responsibilities. The development of an AOCS would affect the operations and responsibilities of the MFCO but is not intended to totally replace these personnel. Although the AOCS would replace most of the actions a MFCO performs during a missile launch, there are many functions accomplished by the MFCO that need human involvement.

The next step in the development process is to formalize the knowledge base into a set of rules in which to build an organized structure. EXMIS is designed as a prototype, proof-of-concept system in order to verify the appropriateness and utility of using an expert system for the AOCS task. Therefore, the design of EXMIS is based upon four separate scenarios that entail a limited portion of the Flight Termination Policies outlined in section 7.3.1 of EWR 127-1. Two of the flight conditions that normally result in flight termination by the MFCO are used to evaluate EXMIS (2:7-4):

a. Valid data shows the launch vehicle has violated established flight safety criteria
b. The performance of the flight vehicle is obviously erratic and the potential exists for the MFCO to lose positive control.

The two conditions above were used as a guideline to develop three of the four flight scenarios involving an erratic flight, that violates established flight criteria, and two uncontrollable flights. The fourth flight scenario is a nominal flight and was used as a baseline. Each of the scenarios is discussed in detail in chapter 5.

4.4 Structure of EXMIS

EXMIS can be broken down into five separate modules plus an Initial States section. The five modules are Control, Input, Analysis, Trends, and Decision. Each module performs a separate and vital function for the overall program. The program code developed for each module and the function it has within EXMIS is discussed in the following paragraphs. The structure designed in this process is by no means the only possible structure available for this expert system, but, for a prototype system it is sufficient.

4.4.1 Initial States

The Initial States section of EXMIS establishes the initial conditions under which the expert system will run. It consists of four deffacts that define the initial facts to be added to the fact-list upon program startup. Each fact will first be displayed and then described. The first deffact is:

```
(deffacts gps-information
  (gps long on)
  (gps lat on)
  (gps alt on)
  (imu yawrate on)
  (imu pitchrate on))
```

This deffact establishes that the GPS receiver and IMU sensors are on and working. For simplification of the task, only minimal GPS and IMU data are simulated and used during the development and testing of EXMIS. Other sources of information can be incorporated in later versions of EXMIS.

The next deffact is:

```
(deffacts destruct-line-and-rate-information
 (longline  120 122 123)
 (latline  20 28 40)
 (altline -10  1000  100000000)
 (yawrate .001)
 (pitchrate .001))
```

This deffact establishes the destruct line, yaw rate, and pitch rate limits that are allowed for the launch vehicle. These limits are represented in a low, nominal, and high format. For instance, the limits for the longitude destruct line are: low, 120^0 West; nominal, 122^0; and high 123^0. The limits are established by the user during pre-launch activities and can be changed at any time before launch.

The next deffact establishes where EXMIS begins:

```
(deffacts cycle-start
 (data-source file)
 (cycle 1))
```

In EXMIS, timing of the system is tracked as a cycle process. When all appropriate functions have been performed by EXMIS, the current cycle ends and the next begins.

The cycle number is also used by EXMIS to establish how many times a particular event has happened. This will be discussed further when we investigate the Decision phase.

The last deffact of the initial section is:

```
(deffacts startup-phase-information
  (phase-after input analysis)
  (phase-after analysis trends)
  (phase-after trends decision)
  (phase-after decision input)
  (phase input))
```

This deffact is important since it establishes the order in which the phases are processed. The phases work with the cycle to control the flow of the program. The phases correspond to the modules and progress in order from Input to Analysis, Analysis to Trends, Trends to Decision, and Decision back to Input were the process begins over.

4.4.2 CONTROL Module

Once all initial information has been input through the Initial State section, EXMIS proceeds to the Control Module. Here we see the first use of the defrule construct to define the rules of the system. Through this module, the program controls the phase changes and updates the cycle number. Without this module, it would not be possible to cycle through each input from the GPS receiver and IMU.

The first defrule construct is:

```
(defrule CONTROL-change-phases
  (declare (salience -10))
  ?phase <- (phase ?current-phase)
```

```
(phase-after ?current-phase ?next-phase)
=>
(retract ?phase)
(assert (phase ?next-phase)))
```

Notice that this rule has the word CONTROL as the first word for the rule name. This is to indicate that this is a rule for controlling execution of the program. Throughout EXMIS other rules will have INPUT, ANALYSIS, TRENDS, or DECISION as the first word of the rule name to indicate their applicable phase. This rule is used to cycle between the four phases.

The next rule in the Control phase is:

```
(defrule DECISION-assert-next-cycle
 (declare (salience -5))
 (phase decision)
 ?f <- (cycle ?cycle)
 =>
 (retract ?f)
 (assert (next-cycle =(+ 1 ?cycle))))
```

This rule is part of the Decision phase and removes the old cycle of execution and asserts a new fact that indicates the next cycle of execution. Notice that a salience value of -5 is given to this rule. This is done so that the rule does not fire until all other rules within the Decision phase have fired.

The last rule in the Control phase is:

```
(defrule INPUT-process-next-cycle
 (phase input)
 ?f <- (next-cycle ?cycle)
 =>
 (retract ?f)
 (assert (cycle ?cycle)))
```

This rule performs an update of the cycle number. As EXMIS moves from cycle to cycle, the fact that represents the current cycle must be updated to the next cycle. The rule effectively removes the *next-cycle* fact during the INPUT phase and asserts the updated cycle fact. This is of vital importance since many of the rules within EXMIS depend upon this fact.

4.4.3 INPUT Module

The next step in EXMIS is to obtain the raw data from the GPS and IMU systems. This is done through the INPUT phase. Although it's possible for EXMIS to read data from actual GPS and IMU systems, for the purposes of this study, the data are provided from a BOOST output file. The INPUT phase consists of only two rules which are described below.

The first rule of the INPUT phase is:

```
(defrule INPUT-get-file-name-from-user
 (phase input)
 (data-source file)
 (not (flt-dat-open))
 =>
 (bind ?flag file-closed)
 (while (eq ?flag file-closed)
  (printout t "What is the name of the file?")
  (bind ?file-name (readline))
  (if (open ?file-name flt-dat "r")
     then (bind ?flag true)))
     (assert (flt-dat-open)))
```

The first process the INPUT phase must handle is to open the applicable BOOST data file. This is accomplished with the above rule. This rule has three simple conditions that must be satisfied. First, the current phase must be the INPUT phase (phase input). This is to ensure that EXMIS has the proper execution timing, as described in section 4.4.1. Second, the data source must be a file (data-source file). In this particular situation it is a BOOST output file. Third, the data file must not have previously been opened by the user (not (flt-dat-open). If these three conditions are satisfied, then the RHS of the rule will open a data file whose name is supplied by the user.

The *while* loop is used in conjunction with the ?flag variable in order to set up a convenient way to insure the file has been opened successfully. If the file has not been opened or if the file name is not a proper file, the user will be repeatedly prompted for the name of a source data file until a suitable one is supplied. Once a file has been successfully opened, the *flt-dat-open* fact is asserted.

The second INPUT phase rule is:

```
(defrule INPUT-read-gps-values-from-file
 (phase input)
 (data-source file)
 (flt-dat-open)
 (cycle ?time)
 =>
 (bind ?long (read flt-dat))
   (if (eq ?long EOF) then (halt))
 (bind ?lat (read flt-dat))
   (if (eq ?lat EOF) then (halt))
 (bind ?alt (read flt-dat))
   (if (eq ?alt EOF) then (halt))
 (bind ?yaw (read flt-dat))
```

```
   (if (eq ?yaw EOF) then (halt))
 (bind ?pitch (read flt-dat))
   (if (eq ?pitch EOF) then (halt))
    (assert (gps-long-value ?long ?time))
    (assert (gps-lat-value ?lat ?time))
    (assert (gps-alt-value ?alt ?time))
    (assert (imu-yaw-value ?yaw ?time))
    (assert (imu-pitch-value ?pitch ?time))
    (assert (data-values-read)))
```

This rule is not as complicated as it might first seem. If the LHS patterns are satisfied, the rule reads all of the data values from the user specified data file until the End-Of-File (EOF) is reached. The BOOST output file is formatted in a manner to support this process. The file has data arranged in five columns pertaining to longitude, latitude, altitude, yaw rate, and pitch rate respectively. The rule **binds** the data from the first column with the variable **?long**. The second column data is bound to the variable **?lat**, and so forth for each of the other data values. The rule also asserts the facts as long, lat, alt, yaw, pitch, and GPS or IMU values with the appropriate variable and cycle time (i.e. gps-long-value ?long ?time). The *data-values-read* fact is asserted to act as a flag indicating that the data file has been read for this cycle.

4.4.4 ANALYSIS Module

This phase of EXMIS gets to the heart of the process. During this module, the values input from the BOOST file are analyzed and compared to the initial values for the destruct lines and rotation rates that were provided in the initial states section. Since analysis rules for longitude, latitude and altitude are similar in format, to decrease redundancy only the

rules pertaining to longitude will be described. There are three separate cases analyzed for each parameter; nominal flight, out-of-bounds-low destruct line, and out-of-bounds-high destruct line. Along these same lines, since rules for yaw and pitch are similar, only those for yaw will be described with two cases analyzed; nominal yaw-rate and high-yaw-rate. The first rule in the ANALYSIS module is:

```
(defrule ANALYSIS-long-nominal-flight
 (phase analysis)
 (cycle ?time)
 (gps-long-value ?long ?time)
 (longline ?long-low-destruct-line ?
           ?long-high-destruct-line )
 (test (and (> ?long ?long-low-destruct-line)
       (< ?long ?long-high-destruct-line)))
 =>
 (printout t "Nominal Longitude" crlf)
 (assert (long-state ?long nominal ?time)))
```

The first pattern ensures that the rule applies during the ANALYSIS phase of EXMIS and the second pattern retrieves the current cycle. The third pattern retrieves the longitude value that was input during the INPUT phase. The INPUT phase has produced one fact of this type for each of the parameters. The fourth pattern retrieves the valid ranges (or destruct lines) for the longitude. As stated previously, these values are in the format of low destruct line, nominal, and high destruct line. A **test** is performed to determine if the value for the variable **?long** is above the low destruct line value and below the high destruct line value. If the test is true, the LHS patterns are all satisfied and the rule fires. EXMIS will print the messeage "Nominal Longitude" to let the user know the missile status. EXMIS will also assert the fact that the state of the missile longitude is nominal

(i.e. long-state ?long nominal ?time). This will become useful during the TRENDS phase. Analysis of the yaw and pitch rate are similar to the analysis performed on the longitude value with just minor differences. The yaw and pitch rate are only tested against the initial high yawrate and high pitchrate given during the initial states section. The yaw and pitch rate are also simulated to be derived from the IMU and not the GPS as with the longitude.

The next rule in the ANALYSIS module is:

```
(defrule ANALYSIS-long-low-destruct-line
 (phase analysis)
 (cycle ?time)
 (gps-long-value ?long ?time)
 (longline ?long-low-destruct-line ? ?)
 (test (<= ?long ?long-low-destruct-line))
 =>
 (printout t "Longitude out of bounds LOW" crlf)
 (assert (long-state ?long long-low-destruct-line ?time)))
```

This rule has much the same look as the previous rule and does incorporate a very similar structure. The first three patterns are identical to the previous rule but the fourth is somewhat different. This pattern retrieves the longitude destruct line values but only names the low destruct line value while making the other two values wild cards with the **?** identifier. The wild card identifier means that these values are not known and can be anything (i.e. wild).

The test performed is also somewhat different. In this case, the **?long** variable is tested against the **?long-low-destruct-line** variable. If the value for **?long** is less than the

value for the other variable, then EXMIS determines that the missile has crossed the low destruct line and the LHS patterns are satisfied. EXMIS prints a message to this affect and the **long-state** is set to **long-low-destruct-line**. Similar rules are written to test the high destruct line values for each parameter.

The next rule involves the yaw rate:

```
(defrule ANALYSIS-yaw-high-destruct-rate
 (phase analysis)
 (cycle ?time)
 (imu-yaw-value ?yaw ?time)
 (yawrate ?high-yaw-destruct)
 (test (>= ?yaw ?high-yaw-destruct))
=>
 (printout t "Yaw Rate is over Max rate allowed" crlf)
 (assert (yaw-state ?yaw high-yaw-destruct
                    ?time)))
```

Once again, the format for this rule is very similar to the previous one. The patterns on the LHS are the same except that the values are taken from IMU data and not GPS. The test performed checks the **?yaw** variable value against the **yawrate** value that was input during the initial states section. If the **?yaw** value is higher than the **yawrate** value EXMIS determines that the rate is outside of established allowable limits. The LHS patterns are satisfied and EXMIS prints the "Yaw Rate is over Max rate allowed" warning. The yaw-state **high-yaw-destruct** is asserted. Information gained within the ANALYSIS phase is passed on to the TRENDS phase. Of particular importance to the TRENDS phase is the *state* information, such as the long-state or lat-state, that will be used to determine if a trend is occurring.

4.4.5 TRENDS Module

As the name implies, the TRENDS module performs the task of detecting trends in the behavior of the missile flight. This is accomplished by first defining some initial facts through the deffacts construct shown below:

```
(deffacts start-trend-analysis
  (long-trend 0 unknown 0 0)
  (lat-trend 0 unknown 0 0)
  (alt-trend 0 unknown 0 0)
  (yaw-trend 0 unknown 0 0)
  (pitch-trend 0 unknown 0 0))
```

The rules which update the trend information will depend upon the **long-trend** fact, and the facts for the other parameters, that exist within the fact-list. The starting state for each parameter is indicated by the word *unknown* to indicate that the trend information is not valid at this time. The other values for the long-trend fact are set to zero and will also be updated through the TRENDS phase.

As with the previous modules, the rules developed for the TRENDS module follow a similar format for each of the parameters. Once again, only the rules for the longitude will be used to describe the TRENDS phase. There are, however, two distinct rule types used within the TRENDS phase. One is used to monitor a trend that has not changed since the last cycle and the other will monitor a trend that has changed since the last cycle.

The first rule for the TRENDS module is:

```
(defrule TRENDS-Long-state-has-not-changed
 (phase trends)
 (cycle ?time)
 ?long-cycle <- (long-trend ? ?lngstate ?start-cycle ?end-cycle)
 (test (= ?end-cycle (- ?time 1)))
 (long-state ?long ?lngstate ?time)
 =>
 (retract ?long-cycle)
 (assert (long-trend ?long ?lngstate ?start-cycle ?time)))
```

As this rule implies, it is used to determine if the state of the missile has changed since the last cycle. The first two patterns establish both the correct phase and cycle. The next pattern and the test that follows, are used to find the **long-trend** fact from the previous cycle. The next pattern checks that the state from the previous cycle is the same as the state for the current cycle. The value for the **?lngstate** variable and the **?end-cycle** time are updated.

The second rule performs the opposite function:

```
(defrule TRENDS-Long-state-has-changed
 (phase trends)
 (cycle ?time)
 ?long-cycle <- (long-trend ? ?lngstate ?start-cycle ?end-cycle)
 (test (= ?end-cycle (- ?time 1)))
 (long-state ?long ?new-long-state&~?lngstate ?time)
 =>
 (retract ?long-cycle)
 (assert (long-trend ?long ?new-long-state ?time ?time)))
```

The first four patterns are the same as the previous rule. The conditional pattern **?new-long-state&~lngstate** performs exactly the opposite task as the pattern used in the previous rule. Here, the pattern checks to make sure the state has changed from the last cycle. Since the state has changed, the **?lngstate** variable is updated to the **?new-long-state** and both the **?start-cycle time** and **?end-cycle** time are as well.

4.4.6 DECISION Module

The last module for EXMIS, but certainly not the least, is the DECISION module. The DECISION module takes the state of the missile, established during the TRENDS phase, and determines whether the missile has violated any pre-defined destruct criteria. The process involves two separate steps. The first step is to determine if the missile is outside the destruct criteria and, if so, issue a warning to that effect. The second step is to determine if the missile has been outside the destruct criteria for an established period of cycles and, if this is the case, issue a self-destruct command. The first part of the DECISION module is:

```
(deffacts trend-limits
 (trend-long 6)
 (trend-lat  6)
 (trend-alt  6)
 (trend-yaw  4)
 (trend-pitch 4))
```

This deffact is defined by the user and establishes a period of cycles in which the missile can be outside the preidefined destruct criteria. This is done to allow for the possibility of an erroneous datum point that could be outside the criteria but does not reflect the

actual situation of the missile. The number of cycles can be increased or decreased depending on the level of constraint the user desires.

The rules used within the DECISION module are of two basic designs. The first rule uses a test pattern to check if the missile has been outside the destruct criteria for a period less than that established in the deffacts construct. In this case, a warning is issued that notifies the user that the missile is outside the established destruct criteria. The second rule is similar to the first with the exception that the test determines if the missile state has been outside the established criteria for a period equal to or greater than the period defined in the deffact. Once again, to avoid duplication only the rules for the longitude parameter are described.

The first rule is:

```
(defrule DECISION-warn-long-over-bounds
 (phase decision)
 (cycle ?time)
 (long-trend ?long ?lngstate&long-high-destruct-line|long-low-destruct-line
          ?start ?end)
 (trend-long ?long-length)
 (test (< (+ (- ?end ?start) 1) ?long-length))
 =>
 (printout t "Missile has crossed " ?lngstate crlf)
 (printout t "During Cycle  " ?time crlf))
```

The first two patterns establish the phase and cycle number. The third pattern determines whether the missile state **?lngstate** has violated the **long-high-destruct-line** or the **long-low-destruct-line**. The fourth pattern retrieves the **trend-long** fact for the number of cycles the user wants to allow the criteria to be violated by the missile. The fifth pat-

tern is a test pattern, described earlier, that checks if the missile has been outside the destruct criteria for a period less than that established in the deffacts construct. When EXMIS determines this to be the case, a warning message is issued to that effect. The cycle number is included so the user knows when the destruct criteria was violated.

The second rule is:

```
(defrule DECISION-destroy-long-over-bounds
 (phase decision)
 (cycle ?time)
 (long-trend ?long ?lngstate&long-high-destruct-line|long-low-destruct-line
        ?start ?end)
 (trend-long ?long-length)
 (test (>= (+ (- ?end ?start) 1) ?long-length))
 =>
 (printout t "Missile has crossed " ?lngstate " for more than 6 cycles " crlf)
 (printout t "Self Destruct is Initiated during Cycle  " ?time crlf)
 (halt))
```

This rule acts as the complement of the first rule. The test pattern now checks that if missile state **?lngstate** has violated the **long-high-destruct-line** or the **long-low-destruct-line** for a period equal to or longer than the established period. If this rule fires, EXMIS will print a message that the missile has crossed the destruct criteria for more than the allowed cycles and will subsequently issue a self-destruct order to the AOCS.

The last rule is basically a cleanup step where all of the flags and values are retracted to allow for the next cycle to begin.

```
(defrule DECISION-remove-old-values-and-flags
 (phase decision)
 (cycle ?time)
 (or ?f <- (gps-long-value ? ~?time)
    ?f <- (gps-lat-value ? ~?time)
```

```
   ?f <- (gps-alt-value ? ~?time)
   ?f <- (imu-yaw-value ? ~?time)
   ?f <- (imu-pitch-value ? ~?time)
   ?f <- (long-state ? ? ~?time)
   ?f <- (lat-state ? ? ~?time)
   ?f <- (alt-state ? ? ~?time)
   ?f <- (yaw-state ? ? ~?time)
   ?f <- (pitch-state ? ? ~?time)
   ?f <- (flt-dat-read))
 =>
 (retract ?f))
```

4.5 Summary

By using a module type structure to develop EXMIS the flow of the program is easily understood. This enhances EXMIS' capability to be customized by future users. Although the EXMIS prototype is developed as a proof-of-concept program only, its adaptability to new environments will make follow-up design work much easier. If an advanced EXMIS system is undertaken in the future, the basic system design developed during this research effort will provide a good base from which to start.

V. Test Results

5.1 Introduction

Having obtained simulation data from the BOOST program and development of the EXMIS prototype complete, the next logical step in this research project is to test EXMIS using the BOOST data. This step will determine if EXMIS can perform the tasks it is designed to perform. As with any computer language or simulation, some adjustments were needed once this testing was under way. The adjustments to EXMIS were necessary to accommodate different scenarios. As stated many times previously, there are four different flight scenarios used to test EXMIS with each scenario being run for 180 seconds.

The process for using BOOST data to test EXMIS was consistent throughout each scenario. To start the process, the CLIPS version 6.0 program is initiated on a personal computer. Under the FILE menu, the Load Constructs command is initiated. A dialog box appears and selection of the *exmis.clp* program is made by the user. The screen will show several lines defining the deffacts and defrules within EXMIS. EXMIS is now loaded into the CLIPS 6.0 program. Under the EXECUTION menu, the Reset and Run commands are selected in order. The user is prompted by EXMIS for the BOOST data file name of the particular flight scenario to be run. Once the user has input a proper file name, EXMIS runs the scenario to the end of the data file or until a safety parameter is violated.

It should be reiterated that conditions for flight termination action can be found in EWR 127-1 section 7.3.1 (2:7-4). Under currently used procedures, the MFCO will follow these guidelines to determine if and when flight termination is initiated. Since EXMIS seeks to emulate the actions of the MFCO, the first two conditions were adopted for developing the four scenarios used to test the EXMIS prototype.

The first scenario establishes a criterion by simulating a nominal flight. The nominal flight gives a baseline from which to describe the other three scenarios. The second scenario uses the flight termination condition where valid data shows the launch vehicle has violated established flight safety criteria. This is interpreted to mean the launch vehicle has violated established destruct lines. Once the vehicle violates these lines, self-destruct is initiated by EXMIS as it would be by the MFCO.

The third and fourth scenarios follow the guidelines for a condition where launch vehicle performance is obviously erratic and the potential exists for the MFCO to lose positive control. These two scenarios are similar in that they simulate an excessive rotation rate around the yaw and pitch axes respectively. In other words, the third scenario simulates the missile tumbling around the yaw axis and the fourth simulates the missile tumbling around the pitch axis. The four scenarios are described separately and in greater detail within this chapter.

5.2 Nominal Flight Scenario

The nominal flight scenario gives a guideline from which the other three scenarios are developed. This scenario was the simplest to construct since parameters for the BOOST data generation were basically in place by the Peacekeeper constant thrust model. Only minor adjustments were needed to obtain the desired simulation conditions. Since the nominal flight scenario is the basis for all other scenarios, the variables changed within the Peacekeeper model for the nominal flight are common to each of the scenarios. Table 4 shows the variables and their values. A complete description of each variable can be found in Appendix A.

Table 4

VARIABLES USED BY THE PEACEKEEPER MODEL (1)

Variable	Value	Purpose
SDELT	= 0.1	Integration time step
PRTBL	= 0.5,180.0	Print interval specified at one point every 0.5 sec for 180 sec.
OMEG	= 1	Flag for rotation of earth at 0.0000729211 rad/sec.
OBLATE	= 1	Flag for oblate earth vs. spherical earth option.
BETA0	= 200.0	Earth-relative launch azimuth from true north.
PHIG0	= 34.7	Launch latitude of Vandenberg AFB, CA.
LAMD0	= 120.6	Launch longitude of Vandenberg AFB,CA.

The BOOST trajectory simulation program made it possible to obtain the data necessary to test the EXMIS prototype. Each BOOST simulation was run for 180 seconds with data printouts every 1/2 second. This accurately simulates data that would be received through use of the GPS and IMU systems. The initial conditions for each scenario are:

rotating earth; oblate earth; predetermined drag curve, thrust, kick angle program, and ignition time for each stage. In conjunction with these parameters, the nominal flight had zero yaw and zero pitch (not counting the predefined pitch rate program).

For all scenarios tested during this research project, the longitude destruct line has been set with an upper value of 123^{0} and a lower value of 120^{0}. The latitude destruct line has an upper value of 36.5^{0} and a lower value of 30.5^{0}. The maximum yaw and pitch rate are both set at 0.05236 rad/sec or just over 3 deg/sec. These parameters have been chosen for testing purposes only and are not known to be parameters used during actual launch activities. The altitude portion of the EXMIS program is not used during this research project but is included with EXMIS in order to facilitate future development of the EXMIS prototype.

The nominal flight scenario is designed to simulate a missile flying within a predetermined launch corridor bounded on each side by destruct lines. The destruct lines are determined during pre-launch activities. EXMIS can be modified by the user to establish new destruct lines for different flight scenarios. Figure 6 gives a good visual interpretation of the launch trajectory and destruct lines for the nominal flight scenario.

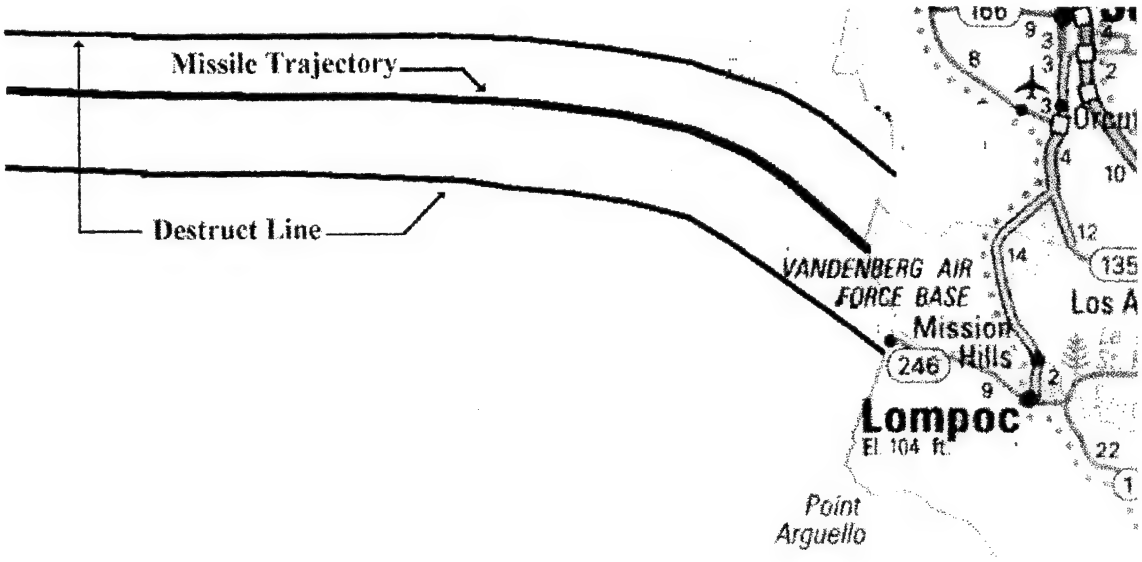

Figure 6 Launch Trajectory from Vandenberg AFB (15)

In order to test EXMIS, BOOST output data is generated by running the Peacekeeper constant thrust model for 180 second. During this time, BOOST simulates a nominal flight for the Peacekeeper ICBM. Although BOOST is capable of outputting a variety of information only the longitude, latitude, altitude, yawrate, and pitchrate are needed to test EXMIS. These parameters are saved within a text file named *pkprnom.txt*.

EXMIS is initiated within the CLIPS program to start the testing phase of the research project. When EXMIS prompts the user for the name of the file to be used, *pkprnom.txt* is entered. Under this scenario, EXMIS is considered to have run successfully when no warnings or self-destruct commands are initiated. The last two pages of the EXMIS output for this scenario can be found in Appendix D. As can be seen from this output, EXMIS does indeed run successfully.

5.3 Errant Missile Scenario

The scenario for an errant missile is very much like the scenario for a nominal flight. The difference between the two is that in the latter case the missile travels within a predefined flight trajectory. With the errant missile case, the launch vehicle develops a small rotation rate in the yaw direction that propels the missile off its predefined trajectory. Since predetermined destruct lines are in place to ensure the safety of personnel and material in the launch range, the situation becomes an emergency. Once the missile crosses a destruct line, EXMIS detects the situation and makes a decision to initiate the self-destruct command. As designed, EXMIS emulates the actions of a human operator, specifically the MFCO in this situation. Figure 7 graphically depicts this situation.

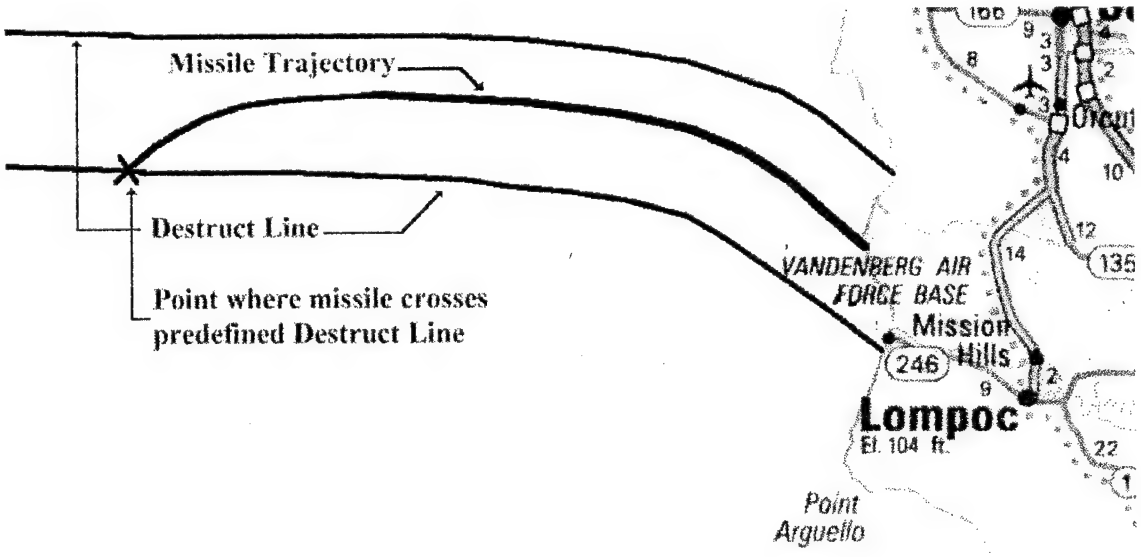

Figure 7 Errant Missile Trajectory from Vandenberg AFB (15)

The introduction of a yaw rate can be caused by many factors. It is assumed, for this simulation, that the adverse yaw experienced by the missile is caused by a malalignment thrust (6:202). This force is sufficient enough to affect the predefined trajectory of a missile. The malalignment thrust can be caused by manufacturing inaccuracies in the structure of a solid propellant rocket or by abnormal fuel flow in a liquid fueled rocket. These inaccuracies can be hard to detect and many times occur without warning. Unless there is an attitude control device located on the missile that is capable of correcting the effects of the adverse yaw, not much can be done by the operator.

The variable used to simulate this effect in the BOOST program is the PSICV variable. With this variable the user can specify a yaw rate for a designated period of time thus causing the missile to deviate from it's intended course. For this simulation, a yaw program is specified that causes a yaw rate of 0.052359 rad or approximately 3 degrees. Under this situation, EXMIS detects if the missile deviates from the path far enough to cross either destruct line. When the missile has crossed the destruct line a warning is issued. The last two pages of the EXMIS output are in Appendix E.

5.4 High Yawrate Scenario

To fully test the capabilities of the EXMIS program, a simulation incorporating a high rotation rate in the yaw direction is necessary. This scenario will test EXMIS' capability to detect abnormal flight characteristics and its ability to provide the appropriate response in this situation. The simulation involves running the EXMIS program in the

CLIPS environment and providing the file name of the BOOST scenario that simulates a missile with a high yawrate. The file name is once again provided by the user when prompted by EXMIS and is named *pkpryaw.txt*. The simulation data is obtained through the use of the PSICV variable, the same variable used in the scenario described in the previous section. This variable is extremely useful in this situation since a yaw program can be defined at any time during the flight of the missile being modeled. Once the start time, stop time, and amount of yaw are determined, it is a simple matter of altering the PSICV variable in the appropriate manner.

To simulate an event that could actually occur doing a missile's flight, the scenario is developed in a manner consistent with a real, physical predicament. There are several phases during a missile's flight where crucial events transpire. In the Peacekeeper model one of those events is the initiation of an attitude hold maneuver. It is at this point in time, when the missile's thrust changes, that problems can arise. The attitude hold maneuver occurs at 117.5 seconds into the missile flight. To simulate a problem with the missile during this thrust change, we configure the PSICV so that a yaw rate of 0.087266 rads (or 5 degrees) begins at 120 seconds. In effect we are simulating the missile tumbling out of control around the yaw axis. Because of the complex dynamics involved, this event is not uncommon during launch vehicle flights. Figure 8 provides a rocket diagram.

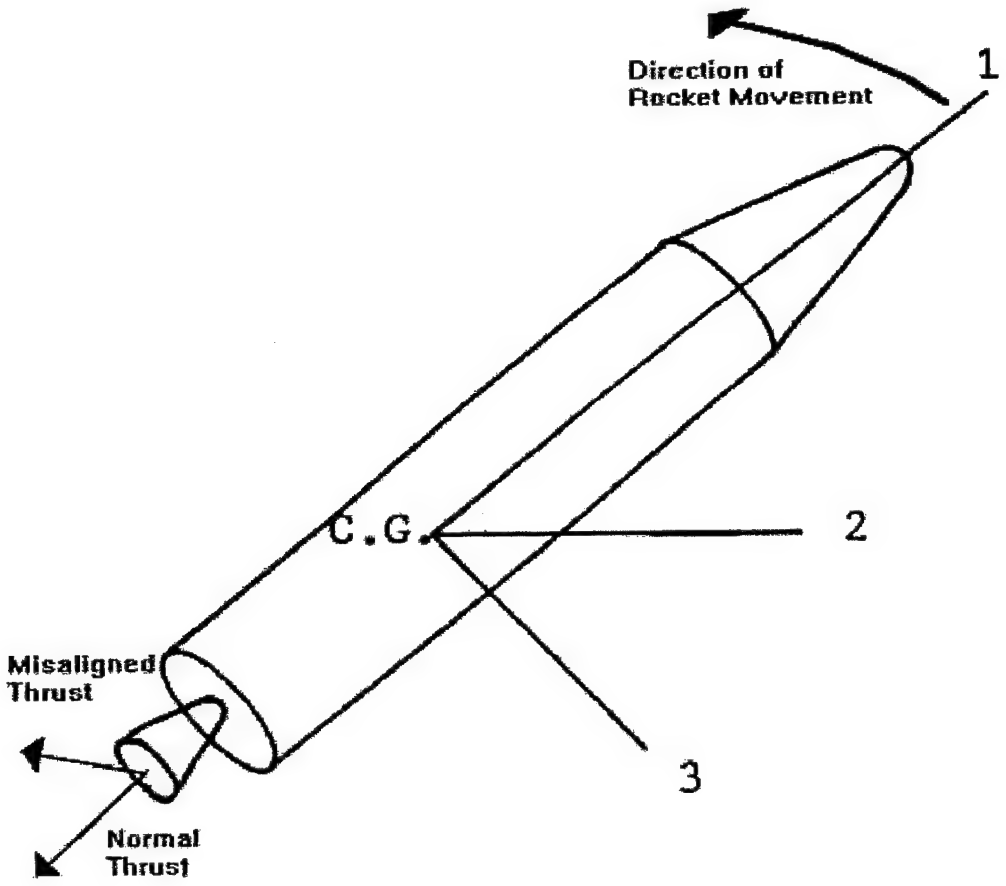

Figure 8 Rocket diagram with misaligned thrust and resulting rocket direction(Yaw Scenario) (1)

5.5 High Pitchrate Scenario

The basis for the high pitchrate scenario is very similar to the yawrate scenario except for the direction in which the missile is rotating. In this situation, the missile has a rotation induced about the pitch axis that causes the missile to rotate in an undesirable manner. Therefore, the missile is acting in a way that is commonly known as tumbling end-over-end. During an actual launch vehicle flight, this tumbling action would cause the MFCO much concern. The MFCO's actions are based upon the guidelines given in EWR 127-1 (2), discussed previously. EXMIS is designed to emulate these actions.

The scenario is developed by building a pitch rate program for BOOST in a similar manner as the yaw rate program was developed. This involves altering a predefined variable named THACV. The THACV variable can be programmed to cause a pitch rate at any given time in the missile's flight. As with the PSICV variable, the start time, stop time, and rate value are needed to define the pitch program.

For the pitchrate scenario, a start time of 120 seconds and a rate of 0.087266 rads (the same values used for the yaw program) were defined. As can be seen, the pitch rate program also coincides with the initiation of the attitude hold maneuver. This was done to give a common reference for the yawrate and pitchrate scenarios and is not intended to indicate that this circumstance is necessary for these two events to occur. Figure 9 shows the simulated misaligned thrust force that causes the pitch rotation condition.

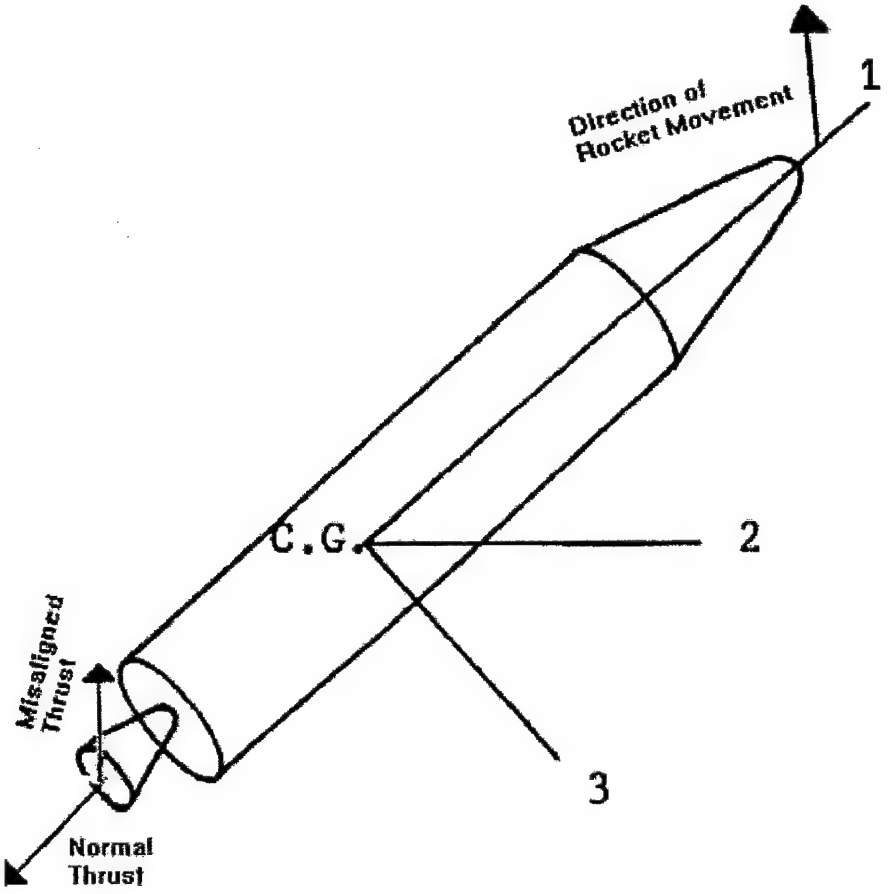

Figure 9 Rocket diagram with misaligned thrust and resulting rocket direction (Pitch) (1)

5.5 Summary

The four scenarios described in this chapter are used to test the accuracy and effectiveness of the EXMIS prototype. Since the scope of this thesis was the development of a prototype expert system, it was felt that testing only four scenarios would give sufficient feedback as to whether EXMIS was a viable system for the AOCS. Every effort was made to develop scenarios that accurately portrayed events which could occur during a launch vehicle's flight. It was felt that developing simulations based upon real-world

situations would lend to greater credibility and acceptance of EXMIS as a useable system. The BOOST program helped achieve this objective by providing useful and accurate simulation data necessary to test the correctness of the EXMIS prototype. The BOOST program is user friendly and is highly recommended for any research involving launch vehicle trajectory simulation.

Although the high pitchrate scenario and high yawrate scenario define only a pitch and yaw respectively, it is easily conceivable that rotations about both axes at the same time is a likely condition. Although it is not a scenario tested during this research project, there is every confidence that EXMIS would detect this situation and perform the appropriate actions necessary to alleviate the problem.

VI. Conclusions and Recommendations

6.1 Conclusions

The main thrust of this research project was the development of the flight termination logic sequence for the AOCS. Although this problem has many possible solutions, it was determined that an expert system could perform this task aptly. In order to accurately depict the environment a launch vehicle experiences while in flight, it became necessary to obtain flight data through means of a simulation. The BOOST trajectory simulation program is well suited for this particular purpose. The learning curve for the BOOST program was very steep and it was a short matter of time before the skills necessary to understand and run the program were achieved.

The CLIPS 6.0 expert system shell was an excellent choice in which to write the desired expert system. Its portability made it possible to work from a PC thus eliminating the time spent sharing hardware with other researchers. Since CLIPS was developed using the C programming language, it has the inherent quality of being easily integrated with other systems. This fact made it possible to develop the EXMIS prototype with the essential ability to be embedded into the AOCS structure. Once in the AOCS, EXMIS can be modified by the user during pre-launch activities in order to cover a wide variety of distinct launch vehicles.

Through this research project, the EXMIS prototype was tested and evaluated utilizing four separate flight scenarios. These scenarios are intended to represent a cross-section of the possible flight scenarios that a launch vehicle is likely to encounter. The complexity of the scenarios are in-line with the objectives of the EXMIS prototype development. EXMIS' responses to these scenarios are an indication of how EXMIS will respond to real-world flight situations. The outcomes from the tests verify that EXMIS will provide task performance on a scale needed for the assigned tasks. As EXMIS and the AOCS are integrated into the operational environment, the users of the system will become more and more familiar with the intricacies of these programs.

6.2 Recommendations

Developing an expert system can be difficult but it is more so when the experts from which the knowledge bases are derived are not easily accessible. This was the case for this research project. The experts for both the AOCS and the current launch procedures were located over 2000 miles away. Communications between the experts and the researcher were limited to telephone conversations. This situation caused problems when trying to obtain even the smallest procedure and self-sufficiency became a necessity. As the research project progressed, more knowledge relating to launch operations was obtained and the role of the AOCS became clearer. This made the task of developing an expert system easier and more intuitive.

To help correct this shortcoming it is recommended that development of an advanced EXMIS prototype be conducted at a location where access to several experts on the AOCS and current launch procedures is readily available. This will alleviate any problems the researcher may have in obtaining information on the subject and will enhance the system development environment. As an advanced prototype progresses, a wider problem domain can be covered until all aspects of the AOCS environment are taken into consideration.

If possible, every effort should be made to incorporate the use of actual GPS and IMU data when testing an advanced version of EXMIS. This will lend credibility to the testing process and could help the developer incorporate real-world parameters into EXMIS. It could also convince skeptics of EXMIS and the AOCS that the system is capable of performing the required task at or above the level of current procedures. The final process should involve in-flight testing of the EXMIS prototype on a limited bases with optional phase-in of the system over a specified period of time.

APPENDIX A: BOOST Variable List

Following is a list of variables that are used within the BOOST program. These variables allow users of BOOST to alter certain aspects of the program to develop unique trajectory simulations. The variables shown here are only those that were altered for use during this research project. For the entire list of variables, the reader should consult the manual "BOOST: A General Six Degree of Freedom Powered Simulation Computer Program" by Anderson (1). A short explanation of each variable and the units of the variable are included.

BETA0----------(deg)
Earth-relative launch azimuth measured from true north. The default value is 31 deg.

BETAF----------(deg)
Orientation azimuth for the AFMTC (Air Force Missile Test Center, Vandenburg AFB, CA) coordinate system

CA2T----------(unitless)
Phase number where you switch from CACV to CACV2. The default is 2.

CA3T----------(unitless)
Phase number where you switch from CACV2 to CACV3.

CACV----------(Mach #, Cd)
This is the first curve for the axial force coefficient. For typical trajectories, this is a good approximation of the drag coefficient, and is exactly the same when the missile is flying a "zero-lift" trajectory. After the first point, the data comes in (Mach Number, Cd) pairs. This curve is usually used to represent drag for the first stage. If the CACV2 curve is used, this is limited to 35 pairs. If this is the only CACV curve used, up to 96 (Mach,Cd) pairs can be entered.
The default curve (also known as the Witt generic drag curve) is:

CACV = -23., 0.0,.2400, 0.4,.2400, 0.6,.2400, 0.7,.2441, 0.8,.2736,
0.9,.3365, 1.0,.4535, 1.1,.4944, 1.2,.5162, 1.3,.5118,

1.4,.5016, 1.6,.4725, 1.8,.4422, 2.0,.4155, 2.5,.3625,
3.0,.3276, 3.5,.3084, 4.0,.3023, 5.0,.2839, 6.0,.2923,
8.0,.2889, 10.0,.2574, 1E10,.2570,

CACV2----------(Mach #,Cd)
This is the second curve for the axial force coefficient. For typical trajectories, this is a good approximation of the drag coefficient, and is exactly the same when the missile is flying a "zero-lift" trajectory. After the first point, the data comes in (Mach Number, Cd) pairs. The curve can be interpreted as follows:
1st index: # of (mach, Cd) pairs
< 0: use linear interpolation
>=0: use highest order interpolation possible (up to cubic)
2,4,...,last even number: mach number
3,5,...,last odd number: coefficient of axial aerodynamic force

This curve is usually used to represent drag for the second stage. If CACV3 is specified, this curve is limited to 35 (Mach,Cd) pairs. If not, then up to 58 pairs are possible.
The default curve (also known as the Witt generic drag curve) is:

CACV2 = -18., 0.0,.5434, 1.8,.5434, 2.2,.4591, 2.70,.3911, 3.05,.3576,
4.3,.2721, 5.7,.2283, 7.1,.2034, 8.25,.1902, 9.50,.1836,
10.8,.1795, 12.1,.1753, 13.5,.1715, 14.70,.1697, 15.70,.1692,
16.9,.1672, 20.0,.1590, 1.E10,.1590,

CACV3----------(Mach #,Cd)
This is the third curve for the axial force coefficient. For typical trajectories, this is a good approximation of the drag coefficient, and is exactly the same when the missile is flying a "zero-lift" trajectory. After the first point, the data comes in (Mach Number, Cd) pairs.
The curve can be interpreted as follows:
1st index: # of (mach, Cd) pairs
< 0: use linear interpolation
>=0: use highest order interpolation possible (up to cubic)
2,4,...,last even number: mach number
3,5,...,last odd number: coefficient of axial aerodynamic force

This curve is usually used to represent drag for either the reentry vehicle on a ballistic missile, or a third stage for a space-launch vehicle. It is limited to 20 (Mach,Cd) pairs, and can only be specified if CACV2 is given.

G0----------(ft/sec^2)
Acceleration of gravity at sea level. Default = 32.174 ft/sec^2.

H0----------(ft)
Launch altitude above sea level. Default = 1 ft.

IIPTB----------(sec,sec)
Table of times for which to calculate the instantaneous impact points.

L1(i)----------(ft)
Length of missile during phase i, where i=1,12.

LAMD0----------(deg)
Launch longitude, where positive is West.

LC----------(ft)
Length of conical section of nose/RV.

OBLATE----------(unitless)
Flag to call oblate earth option.
=0: use spherical earth
=1: use oblate earth model

OMEG,OMEGA----------(rad/sec)
Flag to control rotation of earth.
=0: Non-rotating earth
=1: rotating earth (rate = .00007292115 rad/sec)
This is the same value and serves the same purpose as OMEGA.

PHICV----------(sec,deg/sec)
Table of roll rates as a function of time, where the table is treated as a series of step functions rather than a contiuous curve.

PHIG0----------(deg)
Launch latitude (geodetic), positive North of equator. The default is 45.9 deg N.

PRTBL----------(sec,sec)
<DT>Print interval table. It is interpreted as follows:
1,3,...last odd number: print interval (sec)
2,4,...last odd number: time to end usage of print interval.(sec)
Up to 15 points can be used. The default is :

PSICV----------(deg)
Yaw rate program. The curve is not a continuous function, but instead a series of step functions.The curve can be interpreted as follows:
1: # of points in curve (no interpolation)
2,4,...: Time to begin next yaw rate/program
<0: Use zero side force mode
>0: Use next value as a yaw rate
3,5,...: yaw rate (deg/sec)

RELT----------(ft)
Equatorial radius of oblate earth. Default = 20925738. ft.

SDELT----------(sec)
Integration time step for powered flight. The default is 1 sec.

T1(i)----------(lbf)
Vacuum thrust for phase number i (i = 1,12)

TF1(j)----------(sec)
End of phase time time for phase i (i = 1,12)

THACV----------(sec, deg/sec)
Pitch rate program.

APPENDIX B: EXMIS Program

The following pages provide the complete EXMIS program code. EXMIS is developed using CLIPS (C Language Integrated version 6.0 designed by the NASA/ Johnson Space Center. CLIPS is intended to provide a highly portable, low cost, easily integrated expert sytem shell that can be used to develop complex expert systems. Two sources were invaluable in the creation of EXMIS. One is "Expert Systems: Principals and Programming" by Giarratano and Riley (6). The second is "CLIPS Users Guide: CLIPS Version 6.0" by Giarrantano (5).

```
;;;=======================================================
;;;
;;;   Expert System for the Autonomous Onboard Command System (AOCS)
;;;
;;;   EXMIS:  The Expert System for Missile Self-Destruct
;;;
;;;        This program is desinged to show the utility and
;;;        feasability of using an expert system to monitor
;;;        the flight parameters of a launch vehicle given
;;;        Global Positioning System (GPS) data.  This system
;;;        is capable of issuing a self-destruct command if flight
;;;        parameters fall outside established safety areas
;;;        (i.e. cross destruct lines).
;;;
;;;=======================================================

;;******************
;;* INITIAL STATE ************************************************************
;;******************

(deffacts gps-information
  (gps long on)
  (gps lat on)
  (gps alt on)
  (imu yawrate on)
  (imu pitchrate on))
```

```
(deffacts destruct-line-and-rate-information
 (longline  120 122 123)
 (latline  20 28 40)
 (altline -10  1000  100000000)
 (yawrate .001)
 (pitchrate .001))

(deffacts cycle-start
 (data-source file)
 (cycle 1))

(deffacts startup-phase-information
 (phase-after input analysis)
 (phase-after analysis trends)
 (phase-after trends decision)
 (phase-after decision input)
 (phase input))

;;*****************
;;
;;* CONTROL Module ******************************************************
;;*****************
;;

(defrule CONTROL-change-phases
 (declare (salience -10))
 ?phase <- (phase ?current-phase)
 (phase-after ?current-phase ?next-phase)
 =>
 (retract ?phase)
 (assert (phase ?next-phase)))

(defrule DECISION-assert-next-cycle
 (declare (salience -5))
 (phase decision)
 ?f <- (cycle ?cycle)
 =>
 (retract ?f)
 (assert (next-cycle =(+ 1 ?cycle))))

(defrule INPUT-process-next-cycle
 (phase input)
 ?f <- (next-cycle ?cycle)
 =>
 (retract ?f)
 (assert (cycle ?cycle)))
```

```
;;***************
;;
;;* INPUT Module ************************************************************
;;***************
;;
(defrule INPUT-file-name-from-user
  (phase input)
  (data-source file)
  (not (flt-dat-open))
  =>
  (bind ?flag file-closed)
  (while (eq ?flag file-closed)
    (printout t "What is the name of the file?")
    (bind ?file-name (readline))
    (if (open ?file-name flt-dat "r")
        then (bind ?flag true)))
        (assert (flt-dat-open)))


(defrule INPUT-read-gps-values-from-file
  (phase input)
  (data-source file)
  (flt-dat-open)
  (cycle ?time)
  =>
  (bind ?long (read flt-dat))
    (if (eq ?long EOF) then (halt))
  (bind ?lat (read flt-dat))
    (if (eq ?lat EOF) then (halt))
  (bind ?alt (read flt-dat))
    (if (eq ?alt EOF) then (halt))
  (bind ?yaw (read flt-dat))
    (if (eq ?yaw EOF) then (halt))
  (bind ?pitch (read flt-dat))
    (if (eq ?pitch EOF) then (halt))
     (assert (gps-long-value ?long ?time))
     (assert (gps-lat-value ?lat ?time))
     (assert (gps-alt-value ?alt ?time))
     (assert (imu-yaw-value ?yaw ?time))
     (assert (imu-pitch-value ?pitch ?time))
     (assert (data-values-read)))
```

```
;;*******************
;;
;;* ANALYSIS Module ************************************************
;;*******************
;;

(defrule ANALYSIS-long-nominal-flight
  (phase analysis)
  (cycle ?time)
  (gps-long-value ?long ?time)
  (longline ?long-low-destruct-line ?
            ?long-high-destruct-line )
  (test (and (> ?long ?long-low-destruct-line)
         (< ?long ?long-high-destruct-line)))
  =>
  (printout t "Nominal Longitude" crlf)
  (assert (long-state ?long nominal ?time)))

(defrule ANALYSIS-lat-nominal-flight
  (phase analysis)
  (cycle ?time)
  (gps-lat-value ?lat ?time)
  (latline ?lat-low-destruct-line ?
            ?lat-high-destruct-line )
  (test (and (> ?lat ?lat-low-destruct-line)
         (< ?lat ?lat-high-destruct-line)))
  =>
  (printout t "Nominal Latitude" crlf)
  (assert (lat-state ?lat nominal ?time)))

(defrule ANALYSIS-alt-nominal-flight
  (phase analysis)
  (cycle ?time)
  (gps-alt-value ?alt ?time)
  (altline ?alt-low-destruct-line ?
            ?alt-high-destruct-line )
  (test (and (> ?alt ?alt-low-destruct-line)
         (< ?alt ?alt-high-destruct-line)))
  =>
  (printout t "Nominal Altitude" crlf)
  (assert (alt-state ?alt nominal ?time)))

(defrule ANALYSIS-yaw-nominal-flight
  (phase analysis)
  (cycle ?time)
  (imu-yaw-value ?yaw ?time)
  (yawrate ?high-yaw-destruct)
```

```
 (test (< ?yaw ?high-yaw-destruct))
 =>
 (printout t "Nominal Yaw Rate" crlf)
 (assert (yaw-state ?yaw nominal ?time)))

(defrule ANALYSIS-pitch-nominal-flight
 (phase analysis)
 (cycle ?time)
 (imu-pitch-value ?pitch ?time)
 (pitchrate ?high-pitch-destruct)
 (test (< ?pitch ?high-pitch-destruct))
 =>
 (printout t "Nominal Pitch Rate" crlf)
 (assert (pitch-state ?pitch nominal ?time)))

(defrule ANALYSIS-long-low-destruct-line
 (phase analysis)
 (cycle ?time)
 (gps-long-value ?long ?time)
 (longline ?long-low-destruct-line ? ?)
 (test (<= ?long ?long-low-destruct-line))
 =>
 (printout t "Longitude out of bounds LOW" crlf)
 (assert (long-state ?long long-low-destruct-line ?time)))

(defrule ANALYSIS-lat-low-destruct-line
 (phase analysis)
 (cycle ?time)
 (gps-lat-value ?lat ?time)
 (latline ?lat-low-destruct-line ? ?)
 (test (<= ?lat ?lat-low-destruct-line))
 =>
 (printout t "Latitude out of bounds LOW" crlf)
 (assert (lat-state ?lat lat-low-destruct-line
                    ?time)))

(defrule ANALYSIS-alt-low-destruct-line
 (phase analysis)
 (cycle ?time)
 (gps-alt-value ?alt ?time)
 (altline ?alt-low-destruct-line ? ?)
 (test (<= ?alt ?alt-low-destruct-line))
 =>
 (assert (alt-state ?alt alt-low-destruct-line
                    ?time)))
```

```
(defrule ANALYSIS-long-high-destruct-line
 (phase analysis)
 (cycle ?time)
 (gps-long-value ?long ?time)
 (longline ? ? ?long-high-destruct-line)
 (test (>= ?long ?long-high-destruct-line))
 =>
 (printout t "Longitude out of bounds HIGH" crlf)
 (assert (long-state ?long long-high-destruct-line
                     ?time)))

(defrule ANALYSIS-lat-high-destruct-line
 (phase analysis)
 (cycle ?time)
 (gps-lat-value ?lat ?time)
 (latline ? ? ?lat-high-destruct-line)
 (test (>= ?lat ?lat-high-destruct-line))
 =>
 (printout t "Latitude out of bounds HIGH" crlf)
 (assert (lat-state ?lat lat-high-destruct-line
                    ?time)))

(defrule ANALYSIS-alt-high-destruct-line
 (phase analysis)
 (cycle ?time)
 (gps-alt-value ?alt ?time)
 (altline ? ? ?alt-high-destruct-line)
 (test (>= ?alt ?alt-high-destruct-line))
 =>
 (assert (alt-state ?alt alt-high-destruct-line
                    ?time)))

(defrule ANALYSIS-yaw-high-destruct-rate
 (phase analysis)
 (cycle ?time)
 (imu-yaw-value ?yaw ?time)
 (yawrate ?high-yaw-destruct)
 (test (>= ?yaw ?high-yaw-destruct))
 =>
 (printout t "Yaw Rate is over Max rate allowed" crlf)
 (assert (yaw-state ?yaw high-yaw-destruct
                    ?time)))
```

```
(defrule ANALYSIS-pitch-high-destruct-rate
 (phase analysis)
 (cycle ?time)
 (imu-pitch-value ?pitch ?time)
 (pitchrate ? ?high-pitch-destruct)
 (test (>= ?pitch ?high-pitch-destruct))
 =>
 (printout t "Pitch Rate is over Max rate allowed" crlf)
 (assert (pitch-state ?pitch high-pitch-destruct
                      ?time)))


;;*****************
;;
;;* TRENDS Module  ****************************************************
;;*****************
;;

(deffacts start-trend-analysis
 (long-trend 0 unknown 0 0)
 (lat-trend 0 unknown 0 0)
 (alt-trend 0 unknown 0 0)
 (yaw-trend 0 unknown 0 0)
 (pitch-trend 0 unknown 0 0))

(defrule TRENDS-Long-state-has-not-changed
 (phase trends)
 (cycle ?time)
 ?long-cycle <- (long-trend ? ?lngstate ?start-cycle ?end-cycle)
 (test (= ?end-cycle (- ?time 1)))
 (long-state ?long ?lngstate ?time)
 =>
 (retract ?long-cycle)
 (assert (long-trend ?long ?lngstate ?start-cycle ?time)))

(defrule TRENDS-Long-state-has-changed
 (phase trends)
 (cycle ?time)
 ?long-cycle <- (long-trend ? ?lngstate ?start-cycle ?end-cycle)
 (test (= ?end-cycle (- ?time 1)))
 (long-state ?long ?new-long-state&~?lngstate ?time)
 =>
 (retract ?long-cycle)
 (assert (long-trend ?long ?new-long-state ?time ?time)))

(defrule TRENDS-Lat-state-has-not-changed
 (phase trends)
```

```
 (cycle ?time)
 ?lat-cycle <- (lat-trend ? ?ltstate ?start-cycle ?end-cycle)
 (test (= ?end-cycle (- ?time 1)))
 (lat-state ?lat ?ltstate ?time)
 =>
 (retract ?lat-cycle)
 (assert (lat-trend ?lat ?ltstate ?start-cycle ?time)))

(defrule TRENDS-Lat-state-has-changed
 (phase trends)
 (cycle ?time)
 ?lat-cycle <- (lat-trend ? ?ltstate ?start-cycle ?end-cycle)
 (test (= ?end-cycle (- ?time 1)))
 (lat-state ?lat ?new-lat-state&~?ltstate ?time)
 =>
 (retract ?lat-cycle)
 (assert (lat-trend ?lat ?new-lat-state ?time ?time)))

(defrule TRENDS-Alt-state-has-not-changed
 (phase trends)
 (cycle ?time)
 ?alt-cycle <- (alt-trend ? ?astate ?start-cycle ?end-cycle)
 (test (= ?end-cycle (- ?time 1)))
 (alt-state ?alt ?astate ?time)
 =>
 (retract ?alt-cycle)
 (assert (alt-trend ?alt ?astate ?start-cycle ?time)))

(defrule TRENDS-Alt-state-has-changed
 (phase trends)
 (cycle ?time)
 ?alt-cycle <- (alt-trend ? ?astate ?start-cycle ?end-cycle)
 (test (= ?end-cycle (- ?time 1)))
 (alt-state ?alt ?new-alt-state&~?astate ?time)
 =>
 (retract ?alt-cycle)
 (assert (alt-trend ?alt ?new-alt-state ?time ?time)))

(defrule TRENDS-Yaw-state-has-not-changed
 (phase trends)
 (cycle ?time)
 ?yaw-cycle <- (yaw-trend ? ?ystate ?start-cycle ?end-cycle)
 (test (= ?end-cycle (- ?time 1)))
 (yaw-state ?yaw ?ystate ?time)
 =>
```

```
 (retract ?yaw-cycle)
 (assert (yaw-trend ?yaw ?ystate ?start-cycle ?time)))

(defrule TRENDS-Yaw-state-has-changed
 (phase trends)
 (cycle ?time)
 ?yaw-cycle <- (yaw-trend ? ?ystate ?start-cycle ?end-cycle)
 (test (= ?end-cycle (- ?time 1)))
 (yaw-state ?yaw ?new-yaw-state&~?ystate ?time)
 =>
 (retract ?yaw-cycle)
 (assert (yaw-trend ?yaw ?new-yaw-state ?time ?time)))

(defrule TRENDS-Pitch-state-has-not-changed
 (phase trends)
 (cycle ?time)
 ?pitch-cycle <- (pitch-trend ? ?pstate ?start-cycle ?end-cycle)
 (test (= ?end-cycle (- ?time 1)))
 (pitch-state ?pitch ?pstate ?time)
 =>
 (retract ?pitch-cycle)
 (assert (pitch-trend ?pitch ?pstate ?start-cycle ?time)))

(defrule TRENDS-Pitch-state-has-changed
 (phase trends)
 (cycle ?time)
 ?pitch-cycle <- (pitch-trend ? ?pstate ?start-cycle ?end-cycle)
 (test (= ?end-cycle (- ?time 1)))
 (pitch-state ?pitch ?new-pitch-state&~?pstate ?time)
 =>
 (retract ?pitch-cycle)
 (assert (pitch-trend ?pitch ?new-pitch-state ?time ?time)))

;;*******************
;;* DECISION Module ************************************************
;;*******************

(deffacts trend-limits
 (trend-long 6)
 (trend-lat  6)
 (trend-alt  6)
 (trend-yaw  4)
 (trend-pitch 4))
```

```
(defrule DECISION-warn-long-over-bounds
 (phase decision)
 (cycle ?time)
 (long-trend ?long ?lngstate&long-high-destruct-line|long-low-destruct-line
          ?start ?end)
 (trend-long ?long-length)
 (test (< (+ (- ?end ?start) 1) ?long-length))
 =>
 (printout t "Missile has crossed " ?lngstate crlf)
 (printout t "During Cycle  " ?time crlf))

(defrule DECISION-destroy-long-over-bounds
 (phase decision)
 (cycle ?time)
 (long-trend ?long ?lngstate&long-high-destruct-line|long-low-destruct-line
          ?start ?end)
 (trend-long ?long-length)
 (test (>= (+ (- ?end ?start) 1) ?long-length))
 =>
 (printout t "Missile has crossed " ?lngstate " for more than 6 cycles " crlf)
 (printout t "Self Destruct is Initiated during Cycle  " ?time crlf)
 (halt))

(defrule DECISION-warn-lat-over-bounds
 (phase decision)
 (cycle ?time)
 (lat-trend ?lat ?ltstate&lat-high-destruct-line|lat-low-destruct-line
          ?start ?end)
 (trend-lat ?lat-length)
 (test (< (+ (- ?end ?start) 1) ?lat-length))
 =>
 (printout t "Missile has crossed " ?ltstate crlf)
 (printout t "During Cycle  " ?time crlf))

(defrule DECISION-destroy-lat-over-bounds
 (phase decision)
 (cycle ?time)
 (lat-trend ?lat ?ltstate&lat-high-destruct-line|lat-low-destruct-line
          ?start ?end)
 (trend-lat ?lat-length)
 (test (>= (+ (- ?end ?start) 1) ?lat-length))
 =>
 (printout t "Missile has crossed " ?ltstate " for more than 6 cycles" crlf)
 (printout t "Self Destruct is Initiated during Cycle  " ?time crlf)
 (halt))
```

```
(defrule DECISION-warn-yaw-over-bounds
 (phase decision)
 (cycle ?time)
 (yaw-trend ?yaw ?ystate&high-yaw-destruct
          ?start ?end)
 (trend-yaw ?yaw-length)
 (test (< (+ (- ?end ?start) 1) ?yaw-length))
 =>
 (printout t "Missile has crossed " ?ystate crlf)
 (printout t "During Cycle " ?time crlf))

(defrule DECISION-destroy-yaw-over-rate
 (phase decision)
 (cycle ?time)
 (yaw-trend ?yaw ?ystate&high-yaw-destruct
          ?start ?end)
 (trend-yaw ?yaw-length)
 (test (>= (+ (- ?end ?start) 1) ?yaw-length))
 =>
 (printout t "Missile has crossed " ?ystate " for more than 4 cycles" crlf)
 (printout t "Self Destruct is Initiated during Cycle " ?time crlf)
 (halt))

(defrule DECISION-warn-pitch-over-bounds
 (phase decision)
 (cycle ?time)
 (pitch-trend ?pitch ?pstate&high-pitch-destruct
          ?start ?end)
 (trend-pitch ?pitch-length)
 (test (< (+ (- ?end ?start) 1) ?pitch-length))
 =>
 (printout t "Missile has crossed " ?pstate crlf)
 (printout t "During Cycle  " ?time crlf))

(defrule DECISION-destroy-pitch-over-rate
 (phase decision)
 (cycle ?time)
 (pitch-trend ?pitch ?pstate&high-pitch-destruct
          ?start ?end)
 (trend-pitch ?pitch-length)
 (test (>= (+ (- ?end ?start) 1) ?pitch-length))
 =>
 (printout t "Missile has crossed " ?pstate " for more than 4 cycles" crlf)
 (printout t "Self Destruct is Initiated during Cycle  " ?time crlf)
```

```
 (halt))
(defrule DECISION-print-end-of-cycle
 (declare (salience -1))
 (phase decision)
 (cycle ?time)
 (test (> ?time -1))
 =>
 (printout t "************  Cycle number " ?time " has ended *************" crlf))


(defrule DECISION-remove-old-values-and-flags
 (phase decision)
 (cycle ?time)
 (or ?f <- (gps-long-value ? ~?time)
    ?f <- (gps-lat-value ? ~?time)
    ?f <- (gps-alt-value ? ~?time)
    ?f <- (imu-yaw-value ? ~?time)
    ?f <- (imu-pitch-value ? ~?time)
    ?f <- (long-state ? ? ~?time)
    ?f <- (lat-state ? ? ~?time)
    ?f <- (alt-state ? ? ~?time)
    ?f <- (yaw-state ? ? ~?time)
    ?f <- (pitch-state ? ? ~?time)
    ?f <- (flt-dat-read))
 =>
 (retract ?f))
```

APPENDIX C: Peacekeeper Constant Thrust Model

This appendix shows the Peacekeeper ICBM constant thrust model designed by Mr. Dale Witt for use in the BOOST program. By changing selected variables (see appendix A) it is possible to obtain different flight scenarios for the Peacekeeper ICBM. This particular file is for the nominal flight scenario.

```
   *** Peacekeeper (constant thrust model) ***
   ******* U N C L A S S I F I E D *******

Modeler: Mr. Dale B. Witt, FASTC/TANB DSN 787-2654
Date: June 1993
PBV and RV weights removed to make model unclassified.
LAUNCH ALT 0 FT / TEMP 72 DEG F
PK CONSTANT THRUST MODEL

********* Sabot *** coast before ignition ********

TF1(1) =   1.27,        2.7,
W1(1)  = 195444,         -1.,
T1(1)  = 727652,         0.,
WD1(1) =    0.,          0.,
AC1(1) =    0.,          0.,
D1(1)  =    0.,       7.677,

 ******* Stage 1 ****** Stage 2 ***** Stage 3 ***** Payload ***
TF1(3) =   59.3,         117.5,        187.0,        200.,
W1(3)  =   -1.,          87948.,       27002,        9000.,
T1(3)  =  491778,        275788,       68382,
WD1(3) = 1742.714,       934.962,      224.870,
AC1(3) =   2913,         4704,         3789,
D1(3)  =   -1,           -1,           -1,           -1,

TITLE  =  Jettison shroud (747 lb) at 128.5 seconds
WJTB   =  128.5, 747
```

```
********** Guidance Section ***********
Time (sec)
 0-1.27          : Sabot Launch
 1.27            : Coast',
 2.7             : Powered Vertical Takeoff'
 3.1             : Initiate gravity turn
 117.5           : Initiate attitude hold.

 PSICV = -2., 0.0,0.0, 9999, 0.,
 THACV = -2., -3.1, 0.,  9999, 0.,
 TILTM  = 3.1, RTOL =.01,
 SDELT = 0.1, GAMT = 7.2029856, DELGAM =.55,
 GAMT = 6.8,

*** 1st Stage Drag Curve ***

CACV=-18.,
    0.,.727, .1,.717, .2,.270, .3,.213, .6,.198, .8,.194, .9,.230,
    1.0,.353, 1.08,.408, 1.15,.428, 1.3,.412, 1.5,.382, 2.0,.337,
    2.75,.291, 3.5,.256, 4.5,.222, 5.5,.201, 1000.,.201,

*** 2nd stage drag curve ***

CA2T=4.,
CACV2=-5.,
    0.,.205,  4.52,.205,  5.52,.195,  25.0,.200,  1000,.200,

 CA3T=5.,
 CACV3=-2.,
    0.,.094, 1000, .094,

 PRTBL = .5, 180.0, 1000., 1E10, FFTH=000
 OMEG=1.,OBLATE=1., MAXRAN =0
 H0 = 0.0, BETA0 = 200.0, PHIG0 = 34.7, LAMD0 = 120.6
 HRNTY = 393700.8
 IIPTB = 10.,20.,30.,40.,50.,60.,70.,80.,90.,100.,110.,120.,130.,140
 ITMAX=20
 NOITR=0, ITSIGN=0, MAX=15
 FA1(1)  = 1.1
 INDVAR(1)= GAMT
 DEPVAR(1)= GAMR
 DCBO1(1) = -25.00
 TOL1(1)  = 0.01
********** PeaceKeeper Flight **********
 ENDRUN = 9999.
```

APPENDIX D: Nominal Scenario Output

The following pages were taken from the optional output file produced by the EXMIS prototype. It is evident from the output data that EXMIS does not perform actions to terminate this flight scenario and allowed the simulation to run for the entire time of 180 seconds or 360 cycles. This scenario proves that EXMIS is capable of allowing a nominal flight to continue without interruption.

********** Cycle number 351 has ended **********
Nominal Longitude
Nominal Latitude
Nominal Altitude
Nominal Yaw Rate
Nominal Pitch Rate
********** Cycle number 352 has ended **********
Nominal Longitude
Nominal Latitude
Nominal Altitude
Nominal Yaw Rate
Nominal Pitch Rate
********** Cycle number 353 has ended **********
Nominal Longitude
Nominal Latitude
Nominal Altitude
Nominal Yaw Rate
Nominal Pitch Rate
********** Cycle number 354 has ended **********
Nominal Longitude
Nominal Latitude
Nominal Altitude
Nominal Yaw Rate
Nominal Pitch Rate
********** Cycle number 355 has ended **********
Nominal Longitude
Nominal Latitude
Nominal Altitude
Nominal Yaw Rate

Nominal Pitch Rate
********** Cycle number 356 has ended **********
Nominal Longitude
Nominal Latitude
Nominal Altitude
Nominal Yaw Rate
Nominal Pitch Rate
********** Cycle number 357 has ended **********
Nominal Longitude
Nominal Latitude
Nominal Altitude
Nominal Yaw Rate
Nominal Pitch Rate
********** Cycle number 358 has ended **********
Nominal Longitude
Nominal Latitude
Nominal Altitude
Nominal Yaw Rate
Nominal Pitch Rate
********** Cycle number 359 has ended **********
Nominal Longitude
Nominal Latitude
Nominal Altitude
Nominal Yaw Rate
Nominal Pitch Rate
********** Cycle number 360 has ended **********

APPENDIX E: Errant Missile Output

The following pages were taken from the optional EXMIS output file for the errant missile scenario. Inspection of the output data shows the missile did not cross the destruct line until cycle 299 or at approximately 150 seconds into the flight. EXMIS issues two warning messages, one stating the missile is out of longitude bounds and the other that it has crossed the long-high-destruct-line during cycle 299. If the missile continues on this course for more than 6 cycles (or 3 seconds) EXMIS issues a self-destruct command.

********** Cycle number 296 has ended **********
Nominal Longitude
Nominal Latitude
Nominal Altitude
Nominal Yaw Rate
Nominal Pitch Rate
********** Cycle number 297 has ended **********
Nominal Longitude
Nominal Latitude
Nominal Altitude
Nominal Yaw Rate
Nominal Pitch Rate
********** Cycle number 298 has ended **********
Nominal Longitude
Nominal Latitude
Nominal Altitude
Nominal Yaw Rate
Nominal Pitch Rate
********** Cycle number 299 has ended **********
Longitude out of bounds HIGH
Nominal Latitude
Nominal Altitude
Nominal Yaw Rate
Nominal Pitch Rate
Missile has crossed long-high-destruct-line
During Cycle 300

********** Cycle number 300 has ended **********

Longitude out of bounds HIGH
Nominal Latitude
Nominal Altitude
Nominal Yaw Rate
Nominal Pitch Rate
Missile has crossed long-high-destruct-line
During Cycle 301
********** Cycle number 301 has ended **********
Longitude out of bounds HIGH
Nominal Latitude
Nominal Altitude
Nominal Yaw Rate
Nominal Pitch Rate
Missile has crossed long-high-destruct-line
During Cycle 302
********** Cycle number 302 has ended **********
Longitude out of bounds HIGH
Nominal Latitude
Nominal Altitude
Nominal Yaw Rate
Nominal Pitch Rate
Missile has crossed long-high-destruct-line
During Cycle 303
********** Cycle number 303 has ended **********
Longitude out of bounds HIGH
Nominal Latitude
Nominal Altitude
Nominal Yaw Rate
Nominal Pitch Rate
Missile has crossed long-high-destruct-line
During Cycle 304
********** Cycle number 304 has ended **********
Longitude out of bounds HIGH
Nominal Latitude
Nominal Altitude
Nominal Yaw Rate
Nominal Pitch Rate
Missile has crossed long-high-destruct-line for more than 6 cycles
Self Destruct is Initiated during Cycle 305

APPENDIX F: High Yawrate Output

The following pages were taken from the optional EXMIS output file for the high yawrate scenario. Although the yawrate and pitchrate scenarios are separate, they are similar enough that both may be represented by this output file. During cycle 120, EXMIS detects a yaw rate that is out of the established parameters for the missile flight. Two warnings are issued alerting the user to this fact. If the missile continues at this yaw rate for more than 4 cycles or 2 seconds, EXMIS issues a self-destruct command and terminates the flight.

********** Cycle number 115 has ended **********
Nominal Longitude
Nominal Latitude
Nominal Altitude
Nominal Yaw Rate
Nominal Pitch Rate
********** Cycle number 116 has ended **********
Nominal Longitude
Nominal Latitude
Nominal Altitude
Nominal Yaw Rate
Nominal Pitch Rate
********** Cycle number 117 has ended **********
Nominal Longitude
Nominal Latitude
Nominal Altitude
Nominal Yaw Rate
Nominal Pitch Rate
********** Cycle number 118 has ended **********
Nominal Longitude
Nominal Latitude
Nominal Altitude
Nominal Yaw Rate
Nominal Pitch Rate
********** Cycle number 119 has ended **********
Nominal Longitude
Nominal Latitude

Nominal Altitude
Yaw Rate is over Max rate allowed
Nominal Pitch Rate
Missile has crossed high-yaw-destruct
During Cycle 120
********** Cycle number 120 has ended **********
Nominal Longitude
Nominal Latitude
Nominal Altitude
Yaw Rate is over Max rate allowed
Nominal Pitch Rate
Missile has crossed high-yaw-destruct
During Cycle 121
********** Cycle number 121 has ended **********
Nominal Longitude
Nominal Latitude
Nominal Altitude
Yaw Rate is over Max rate allowed
Nominal Pitch Rate
Missile has crossed high-yaw-destruct
During Cycle 122
********** Cycle number 122 has ended **********
Nominal Longitude
Nominal Latitude
Nominal Altitude
Yaw Rate is over Max rate allowed
Nominal Pitch Rate
Missile has crossed high-yaw-destruct for more than 4 cycles
Self Destruct is Initiated during Cycle 123

Bibliography

1. Anderson, E. Z. *Boost: A General Six Degree of Freedom Powered Flight Trajectory Simulation Computer Program*, April, 1972. TR-72-002, Martin Marietta Corporation, Denver, CO.

2. Department of the Air Force. *Range Safety Requirements*. Eastern and Western Range 127-1. 31 March, 1995.

3. Department of the Air Force, Air Force Space Command, 6595 Test and Evaluation Group. Contract F04703-91-C-0110 with ITT Federal Services Corporation. Vandenberg AFB CA, 19 February 1993.

4. Dumont, A., 10 May 1995. Senior Mission Flight Control Officer, 30th Space Wing, Vandenberg AFB CA. Telephone Interview.

5. Foerst, F., 12 April 1995. Instrumentation Engineer, 30th Range Squadron, Vandenberg AFB CA. Telephone Interview.

6. Gantmakher, F. R. and L. M. Levin. *The Flight of Uncontrolled Rockets.* Oxford, England: Pergamon Press Ltd., 1964.

7. Giarratano, Joseph. *CLIPS Users Guide: CLIPS Version 6.0*, May 28, 1993. JSC-25013, Information Systems Directorate, Software Technology Branch, NASA/Johnson Space Center, Houston TX.

8. Giarratano, J. and Gary Riley. *Expert Systems: Principles and Programming.* Boston: PWS-KENT Publishing Company, 1989.

9. Grimes, Fred. *Spacelift Range Architecture Study: Autonomous Onboard Command System (AOCS)*, 23 March, 1993. ITT Federal Services Corporation.

10. Harrington, James B. " CLIPS as a Knowledge Based Language" *Third Conference on Artificial Intelligence for Space Applications*, NASA Conference Publication 2492.

11. Howlin, K., J. Weissier, K. Krantz. "MOORE: A Prototype Expert System for Diagnosing Spacecraft Problems" *1988 Goddard Conference on Space Applications of Artificial Intelligence*, May,1988. NASA Conference Publication 3009.

12. Luczak, E., K. Gopalakrishnan, D. Zillig. "REDEX: The Ranging Equipment Diagnostic Expert System" *1989 Goddard Conference on Space Applications of Artificial Intelligence*, May, 1989. NASA Conference Publication 3033.

13. Martin, J., S. Oxman. *Building Expert Systems: A Tutorial*. Englewood Cliffs NJ: Prentice Hall, 1988.

14. Math Works Inc, The. *MATLAB: High-Performance Numeric Computation and Visualization Software, Reference Guide*. Natick, MA: The MathWorks, Inc, October 1992.

15. Rand McNally & CO. *1995 Road Atlas & Vacation Guide*. Chicago IL: Rand McNally & CO, 1995.

16. Simmons, C. "ISTAR: Intelligent System for Telemetry Analysis in Real-time" *1994 Goddard Conference on Space Applications of Artificial Intelligence*, May, 1993. NASA Conference Publication 3268.

17. Stockbridge, Samuel E. *Autonomous Vehicle Mission Planning Using AI Techniques*. MS Thesis. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1985.

18. Sutton, George P. *Rocket Propulsion Elements: An Introduction to the Engineering of Rockets* (Sixth Edition). New York: John Wiley and Sons, 1992.

19. TRW, Space and Defense Sector, San Bernardino CA. *Peacekeeper Flight Test and Performance Report*. Report number GT-10PA, August 1993.

20. Van Horn, M. *Understanding Expert Systems: An In-Depth Guide to the New Generation Of "Smart" Computer Software*. New York: Bantam Books, 1986.

21. Walters, J., and others. "Autonomous Power Expert System" *1990 Goddard Conference on Space Applications of Artificial Intelligence*, May, 1990. NASA Conference Publication 3068.

Vita

Captain Kevin D. Benedict was born on 3 October 1962 in Columbia, Missouri. He graduated from Harrisburg High School in 1981. He enlisted in the U.S. Air Force in 1986 and was stationed at the U.S. Air Force Academy, Colorado Springs Colorado as a Ground Radio Communications Operator. He graduated with a Bachelor of Science degree in Electrical Engineering from Colorado Technical College in September 1990. He received his commision on 13 February 1991 upon graduation from Officer Training School. His first assignment was at Whiteman AFB as a missile launch officer for the 510th Missile Squadron. In May 1994, Captian Benedict entered the Graduate School of Engineering, Air Force Institute of Technology. Upon completion of a Masters Degree in Space Operations - Space Systems Engineering, Captain Benedict will be assigned to the 30th Range Squadron at Vandenberg AFB, California as a Range Control Officer.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1995	Master's Thesis

4. TITLE AND SUBTITLE
EXMIS: AN EXPERT SYSTEM FOR THE AUTONOMOUS ONBOARD COMMAND SYSTEM (AOCS)

5. FUNDING NUMBERS

6. AUTHOR(S)
Kevin D. Benedict
Captain, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology
Wright-Patterson AFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GSO/ENY/95D-01

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Mr. Fred Foerst
30 RANS/MAL
826 13th St.
Vandenberg AFB, CA. 93437-5212

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The primary objective of this study is the creation of a prototype expert system called EXMIS(EXpert MIssile System) that performs the flight termination logic sequence of the Autonomous Onboard Command System (AOCS). The AOCS is a proposed system that would take the "man-in-the-loop" out of the self-destruct decision making process and place the entire decision on the launch vehicle. Through the use of a six degree of freedom trajectory program called BOOST, simulated flight data for four different flight scenarios is obtained. The launch vehicle selected for the simulations is the Peacekeeper ICBM. EXMIS is developed using an expert system shell called CLIPS (C Language Integrated Production System) designed at the NASA/Johnson Space Center. CLIPS is a forward-chaining rule-based language that has inferencing and representation capabilities. Once developed, the BOOST simulation data are used to evaluate EXMIS under different scenarios involving nominal, errant, and unstable launch vehicle flight.

14. SUBJECT TERMS
Expert Systems, Autonomous Systems
Missile Trajectory, Artificial Intelligence

15. NUMBER OF PAGES
118

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to ***stay within the lines*** to meet ***optical scanning requirements***.

Block 1. Agency Use Only *(Leave blank)*.

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	**PR**	- Project
G	- Grant	**TA**	- Task
PE	- Program Element	**WU**	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. *(If known)*

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of...; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

- **DOD** - See DoDD 5230.24, "Distribution Statements on Technical Documents."
- **DOE** - See authorities.
- **NASA** - See Handbook NHB 2200.2.
- **NTIS** - Leave blank.

Block 12b. Distribution Code.

- **DOD** - Leave blank.
- **DOE** - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
- **NASA** - Leave blank.
- **NTIS** - Leave blank.

Block 13. Abstract. Include a brief *(Maximum 200 words)* factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code *(NTIS only)*.

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.